MACHINE LEARNING DEPARTMENT

Fast, Cheap and Deep

Scaling with the Parameter Server

Alexander Smola

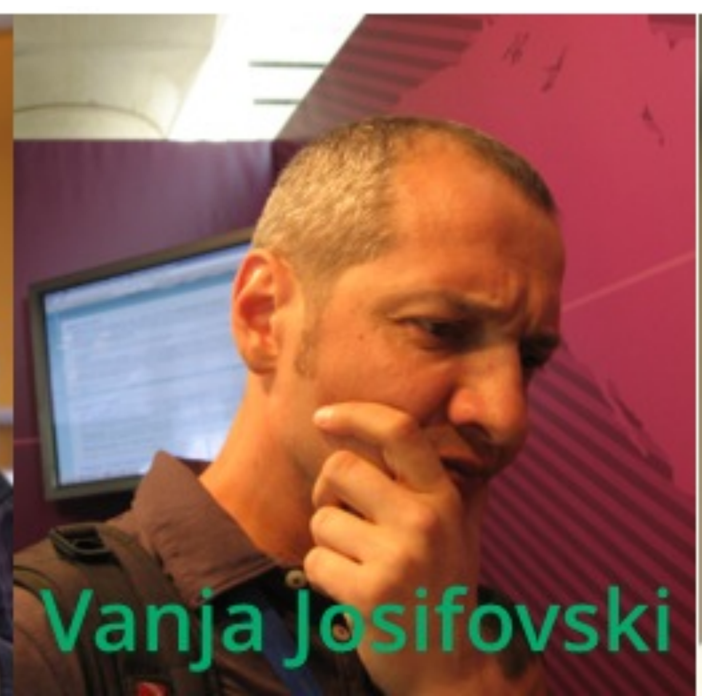
Large Scale Machine Learning 10-605

parameterserver.org

Carnegie Mellon University

Fast, Cheap and Deep

Scaling with the Parameter Server

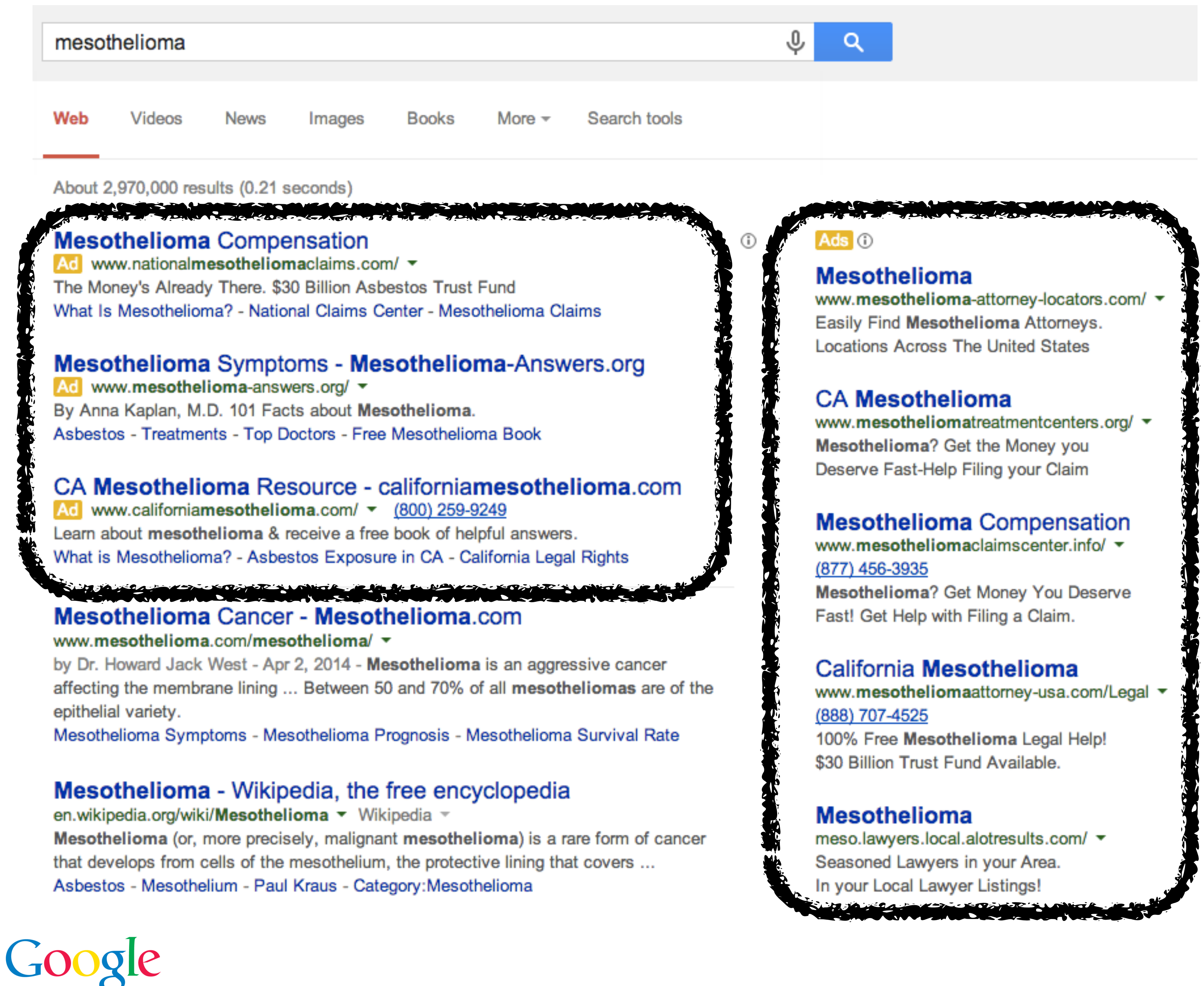


Outline

- **Background**
  Models, hardware
- **Bipartite design**
  Communication, key layout, recovery
- **Efficiency**
  Filters, consistency models
- **Improving the Layout**
  Submodular load balancing
- **Experiments**

Background

Computational Advertising



sponsored search picks position of ad using

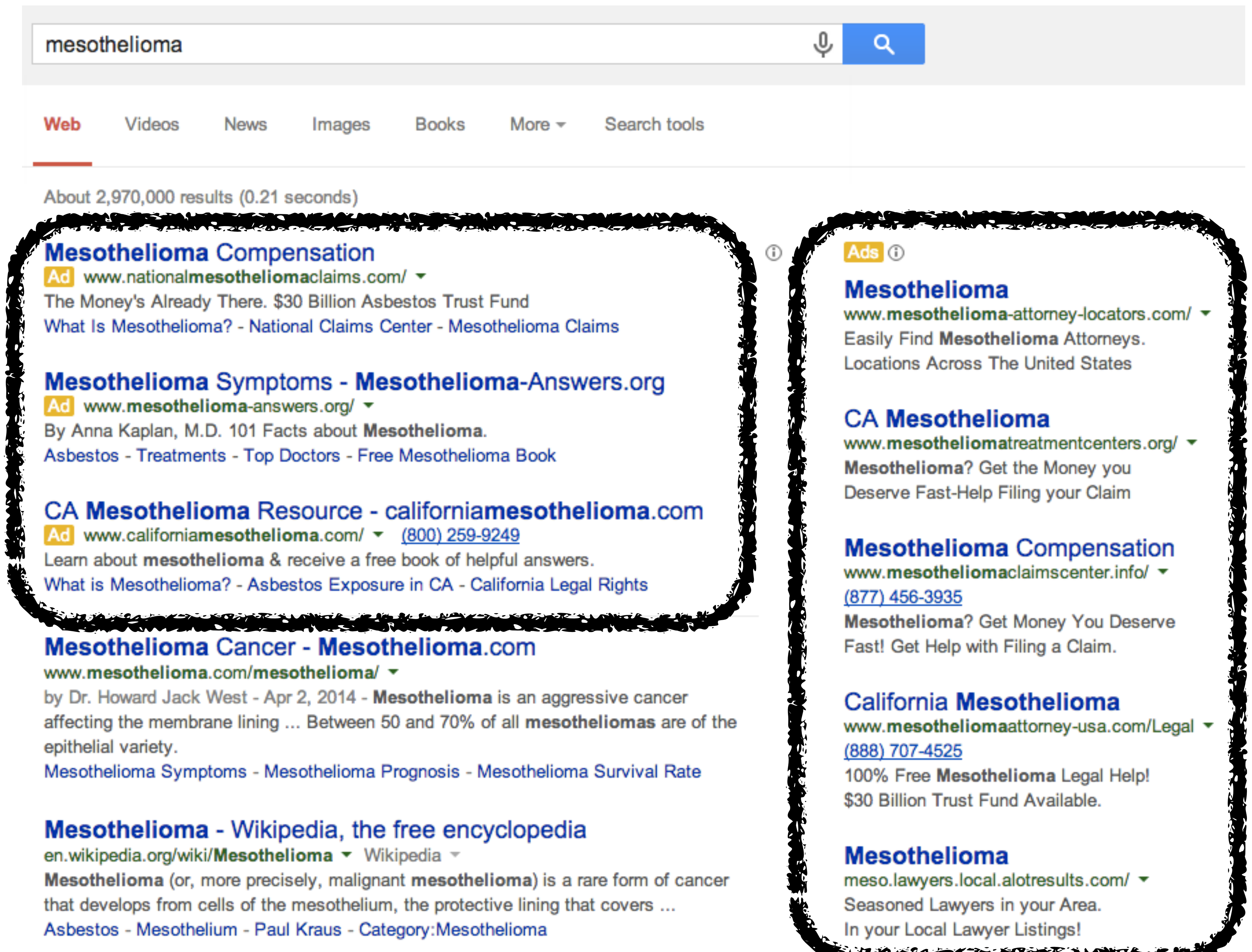
$$p(\text{click}|\text{ad}) \cdot \text{bid}(\text{ad})$$

estimate it

4 million/minute

Carnegie Mellon University

Computational Advertising



Google

sponsored search picks position of ad using

$$p(\text{click}|\text{ad}) \cdot \text{bid}(\text{ad})$$

estimate it

4 million/minute

Carnegie Mellon University

Estimating clicks

- **Logistic regression**

$$p(y|x) = \frac{1}{1 + \exp(-yf(x))}$$

- **Linear function class**

$$f(x) = \langle w, x \rangle$$

we want sparse models for advertising

- **Sparsity prior**

$$\log p(f) = \lambda \|w\|_1 + \text{const.}$$

- **Inference problem**

$$\underset{w}{\text{minimize}} \sum_{i=1}^{m} \log(1 + \exp(-y_i \langle w, x_i \rangle)) + \lambda \|w\|_1$$

Proximal Algorithm

- l_1 norm is non-smooth
- Proximal operator

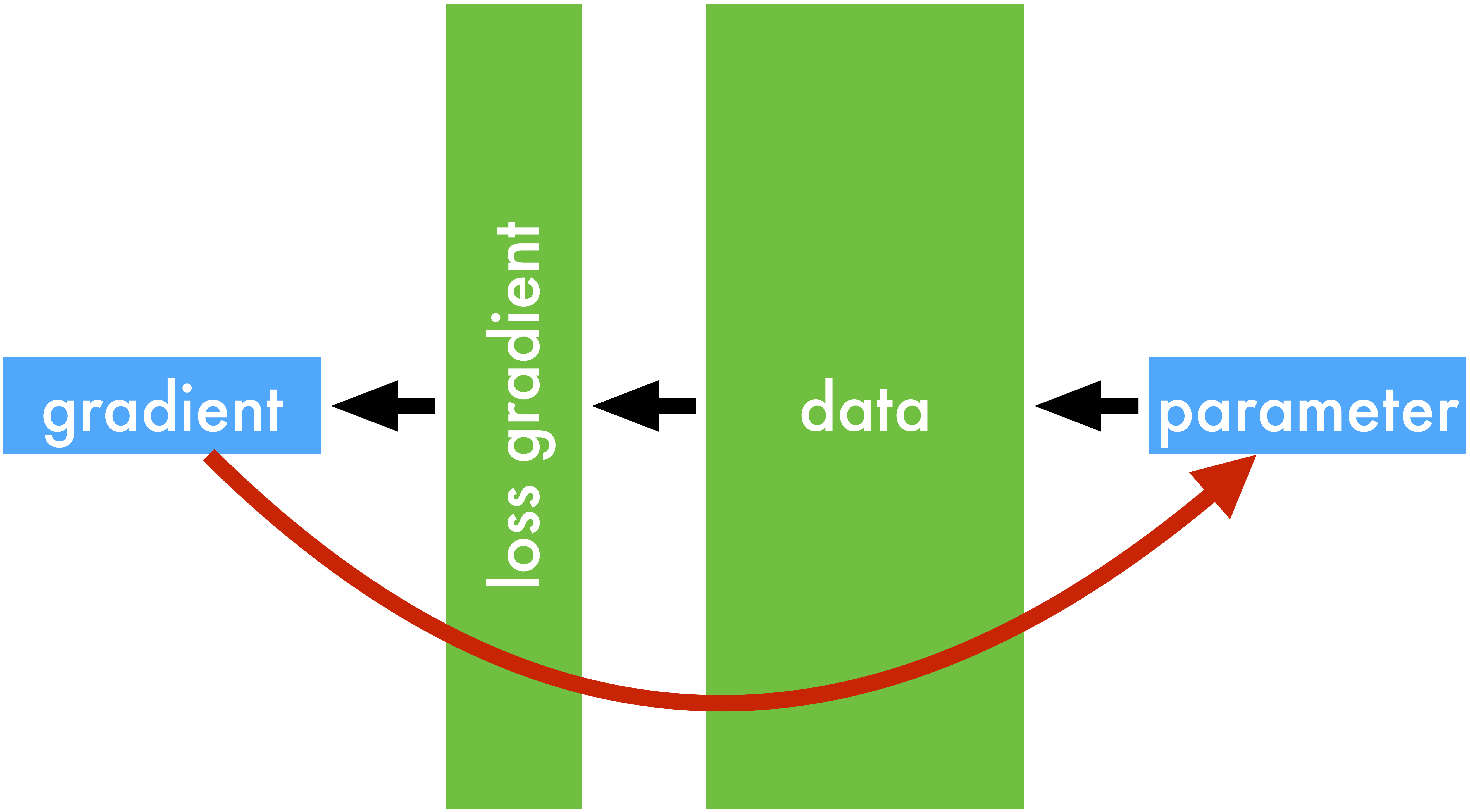
$$\underset{w}{\operatorname{argmin}} \|w\|_1 + \frac{\gamma}{2} \|w - (w_t - \eta g_t)\|^2$$

- Updates for l_1 are

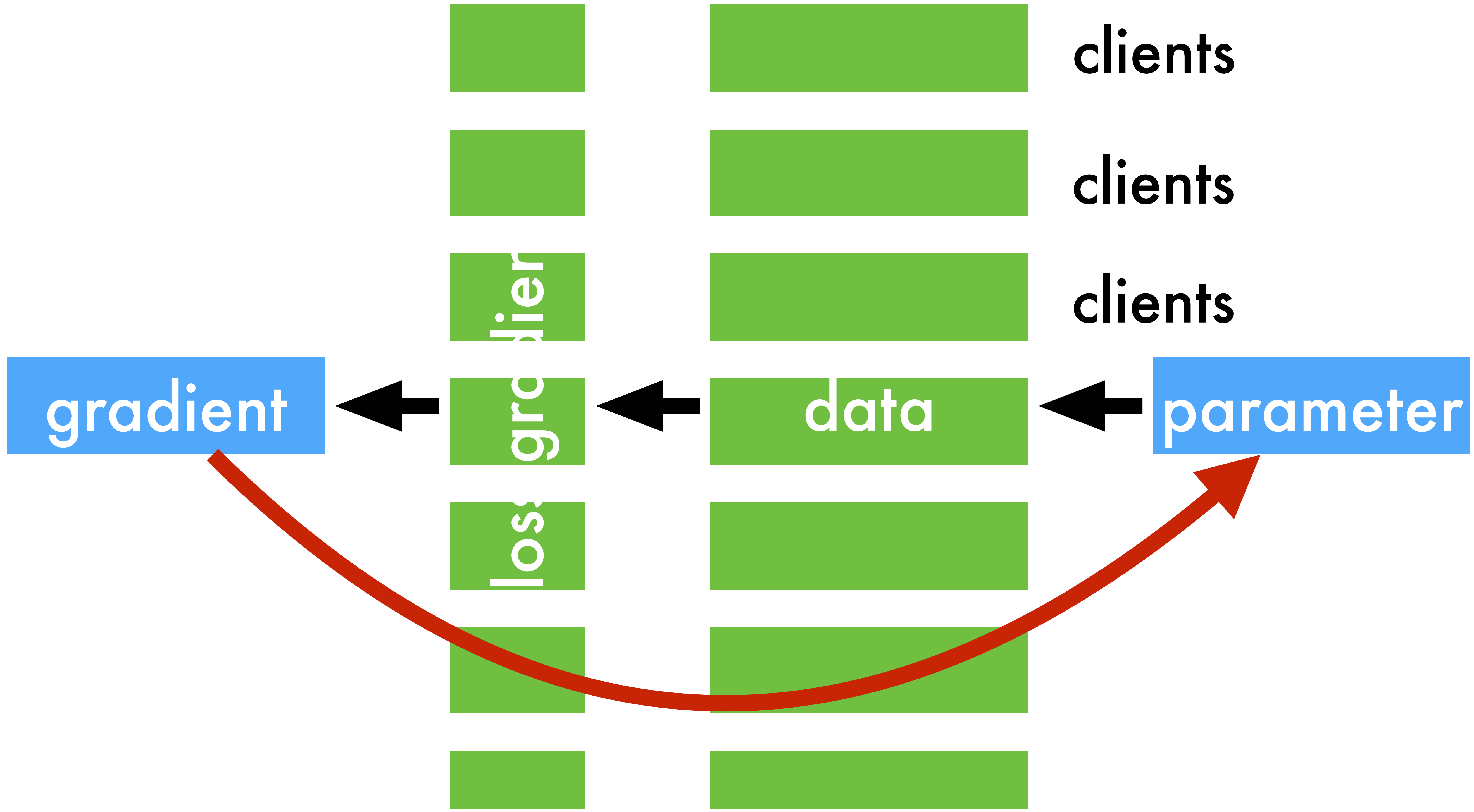
$$w_i \leftarrow \operatorname{sgn}(w_i) \max(0, |w_i| - \epsilon)$$

All steps decompose by coordinates

Data flow



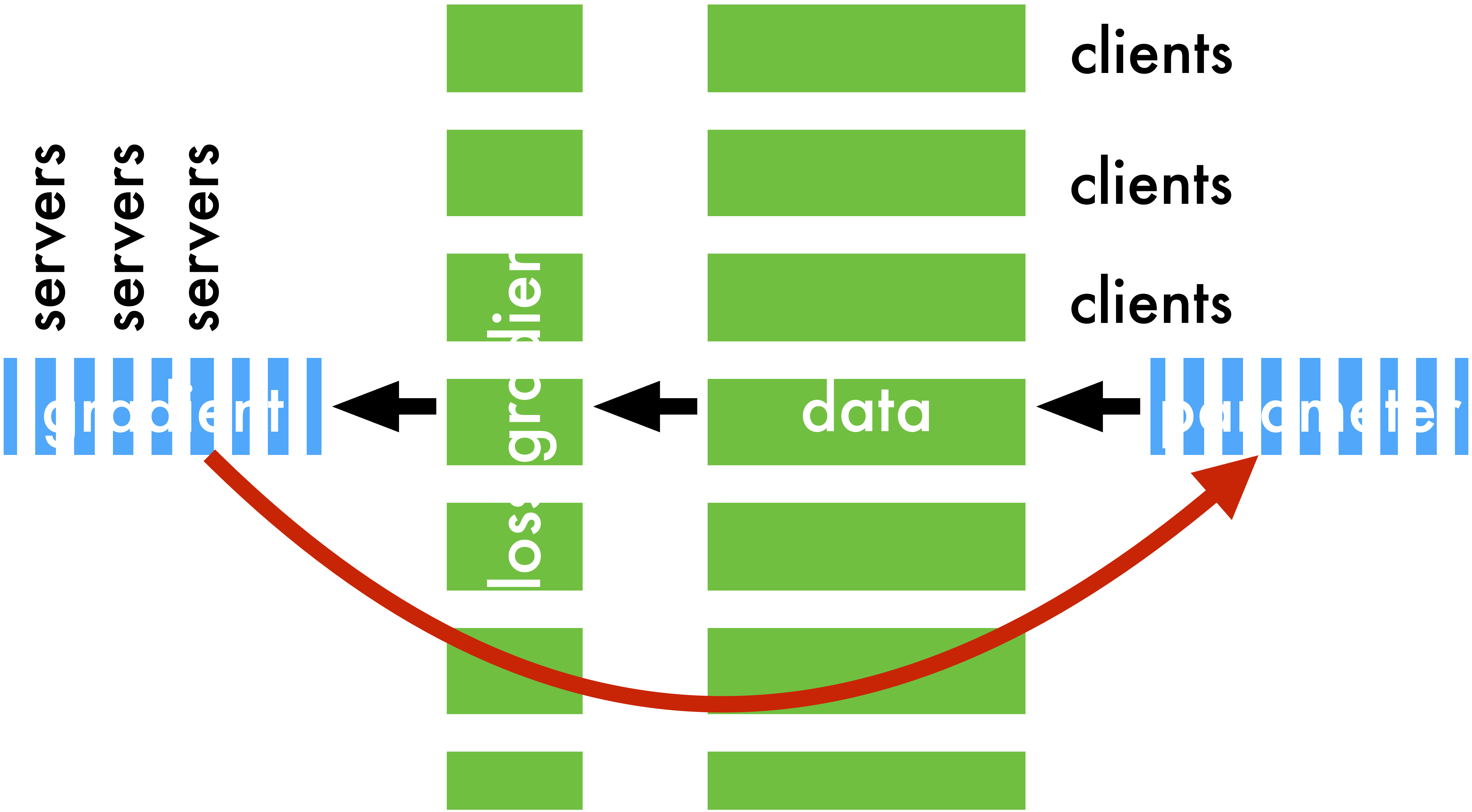
Data flow



Google

Carnegie Mellon University

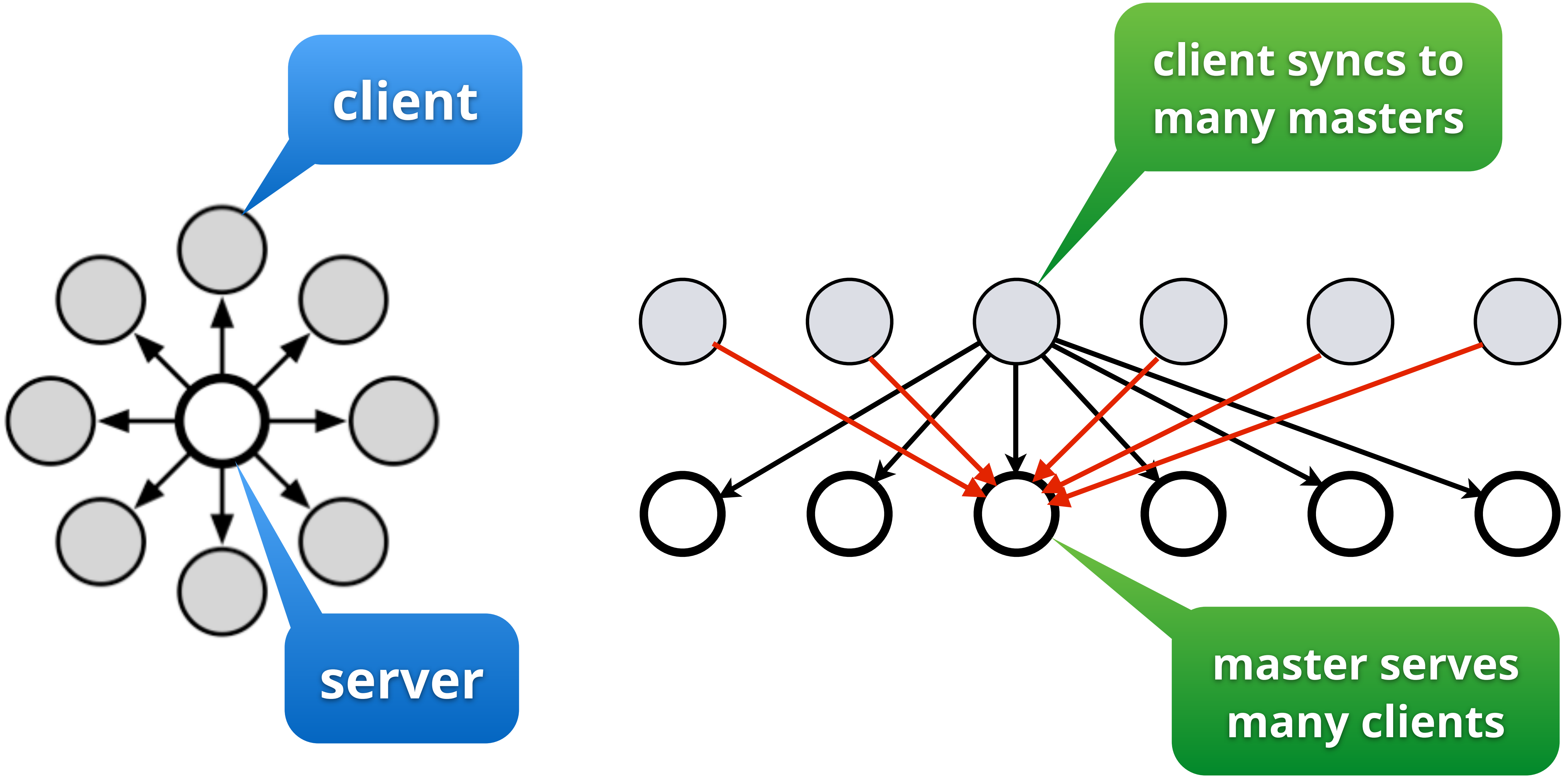
Data flow



Google

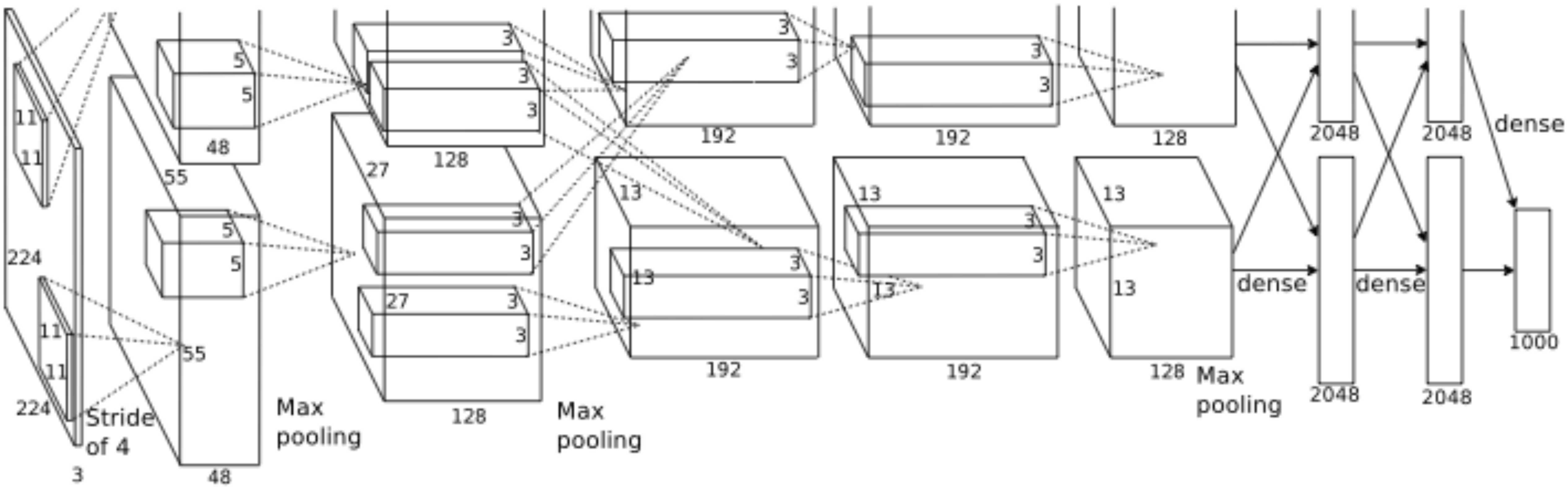
Carnegie Mellon University

Communication pattern



put(keys,values,clock), get(keys,values,clock)

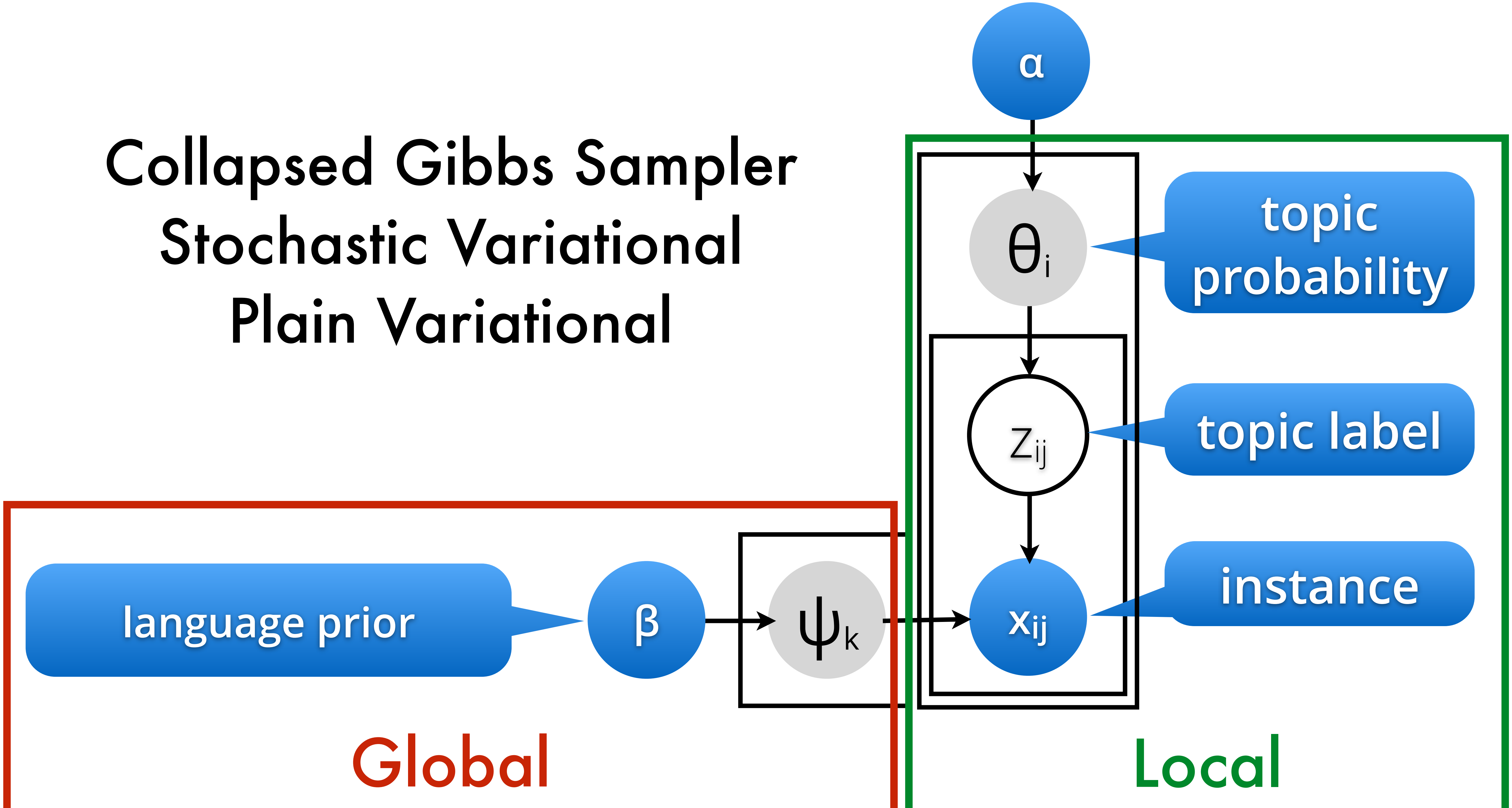
Deep Networks



- Gradients are more structured (groups per layer)
- Hierarchical structure (multi GPU to host to server)

Topic Models

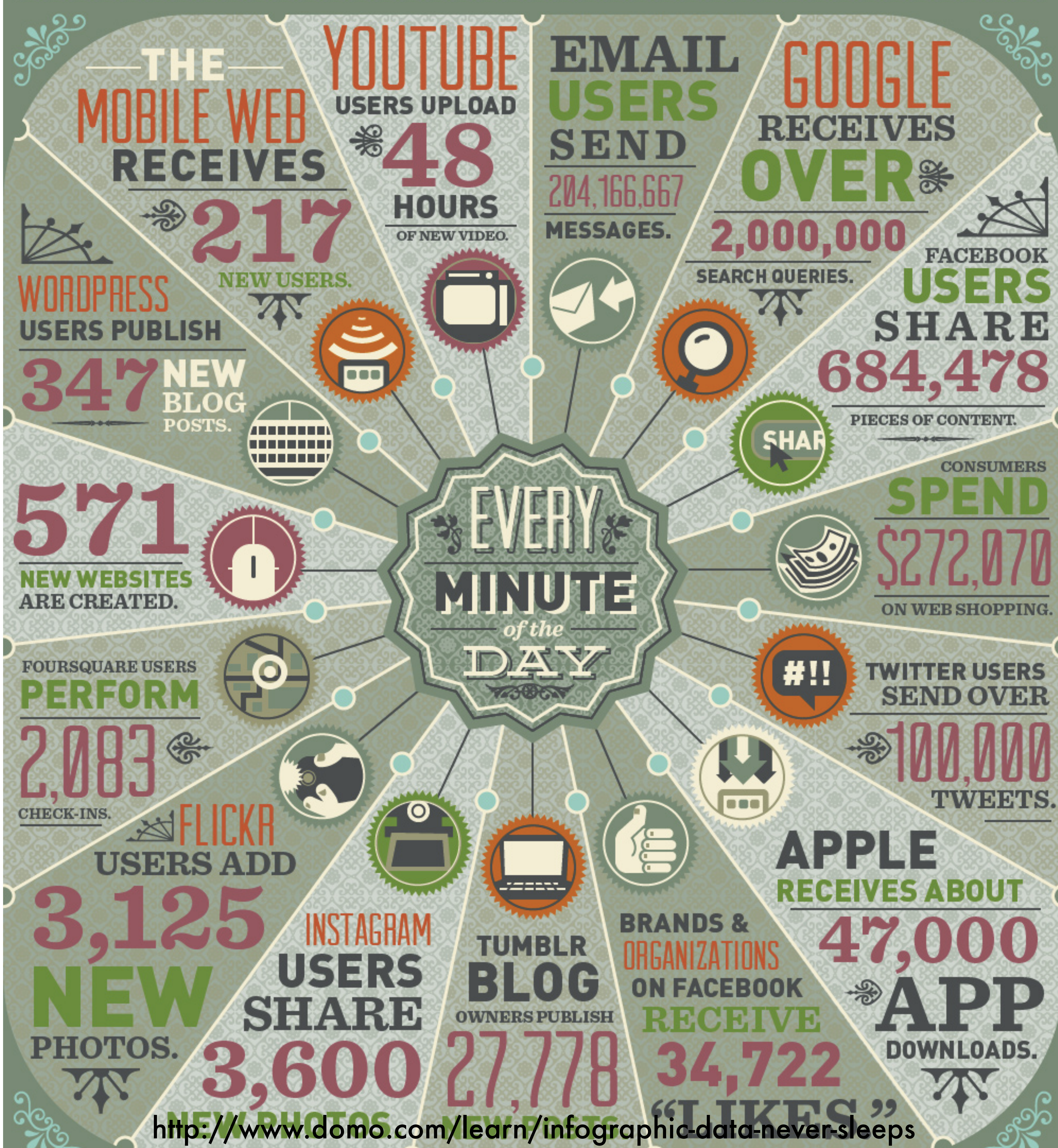
Collapsed Gibbs Sampler
Stochastic Variational
Plain Variational



Machine Learning Redux

- Many models have O(1) blocks of O(n) terms
  (LDA, logistic regression, recommenders, deep)
- More features than what fits into RAM
  (personalized CTR, large inventory, actions, LSTM)
- Local model typically fits into RAM
- Data needs many disks for distribution
- Decouple data processing from aggregation
  (similar idea to MapReduce)
- Sweet spot - optimize for 80% of ML

Data per minute 2012



http://www.domo.com/learn/infographic-data-never-sleeps

Google

Data per minute 2014

We scale

> 100 TB data

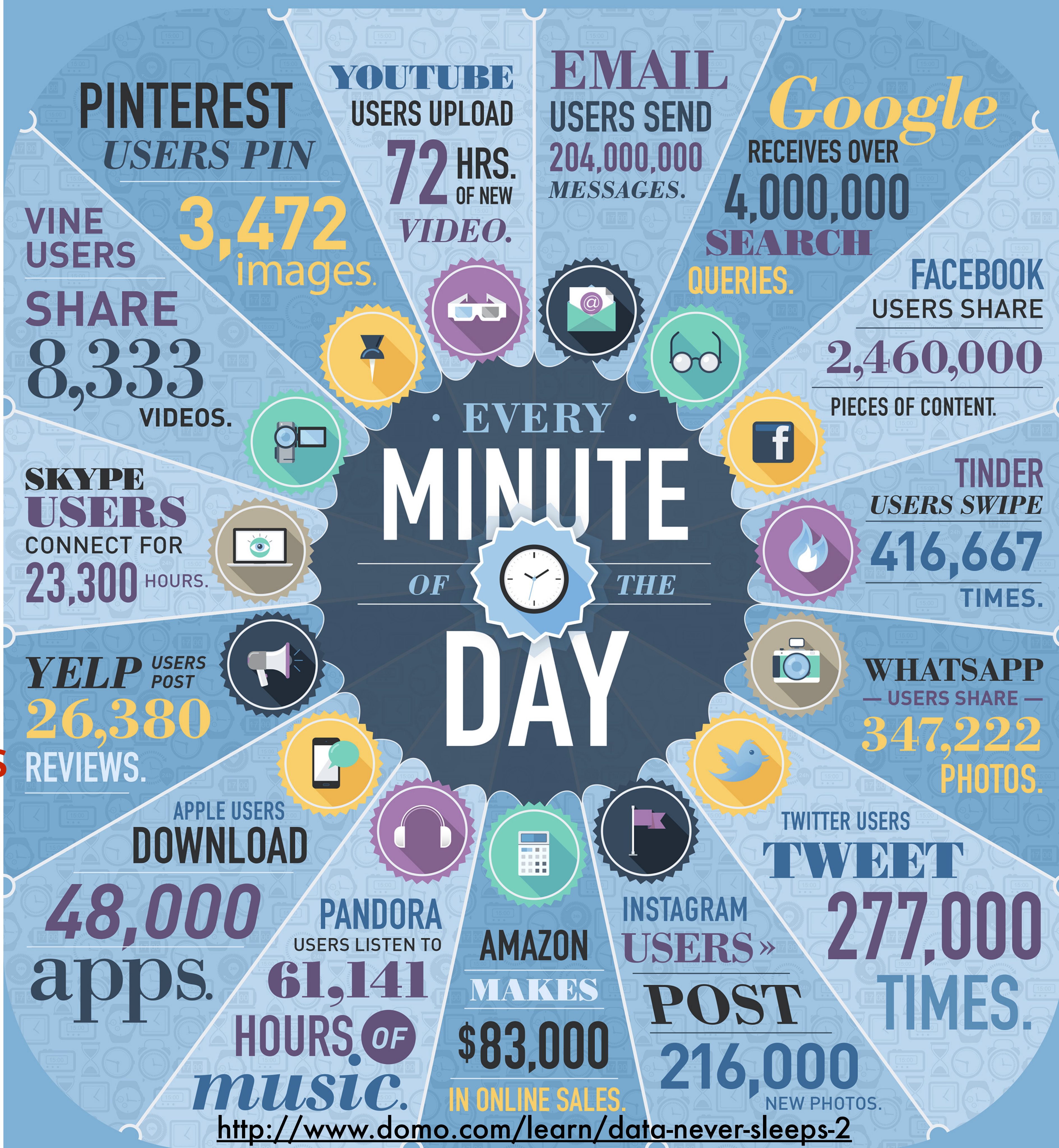
> 1000 machines

> 100B parameters

> 1B inserts/s

> 4B documents

> 2M topics/s



Google

Stuff fails a lot. Deal with it!

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)

~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)

~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)

~1 network rewiring (rolling ~5% of machines down over 2-day span)

~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)

~5 racks go wonky (40-80 machines see 50% packetloss)

~8 network maintenances (4 might cause ~30-minute random connectivity losses)

~12 router reloads (takes out DNS and external vips for a couple minutes)

~3 router failures (have to immediately pull traffic for an hour)

~dozens of minor 30-second blips for dns

~1000 individual machine failures

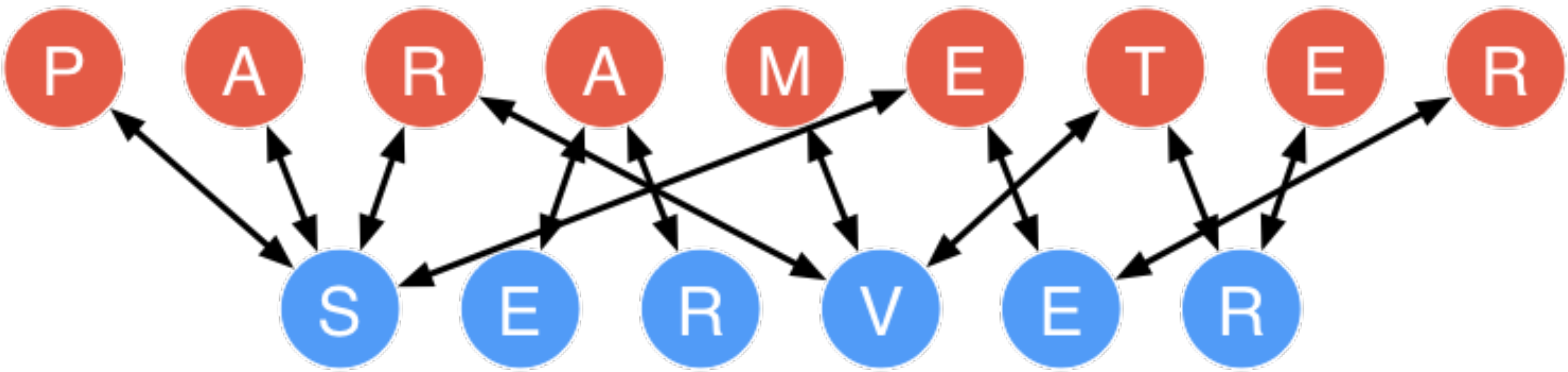
~thousands of hard drive failures

(slide courtesy Jeff Dean)

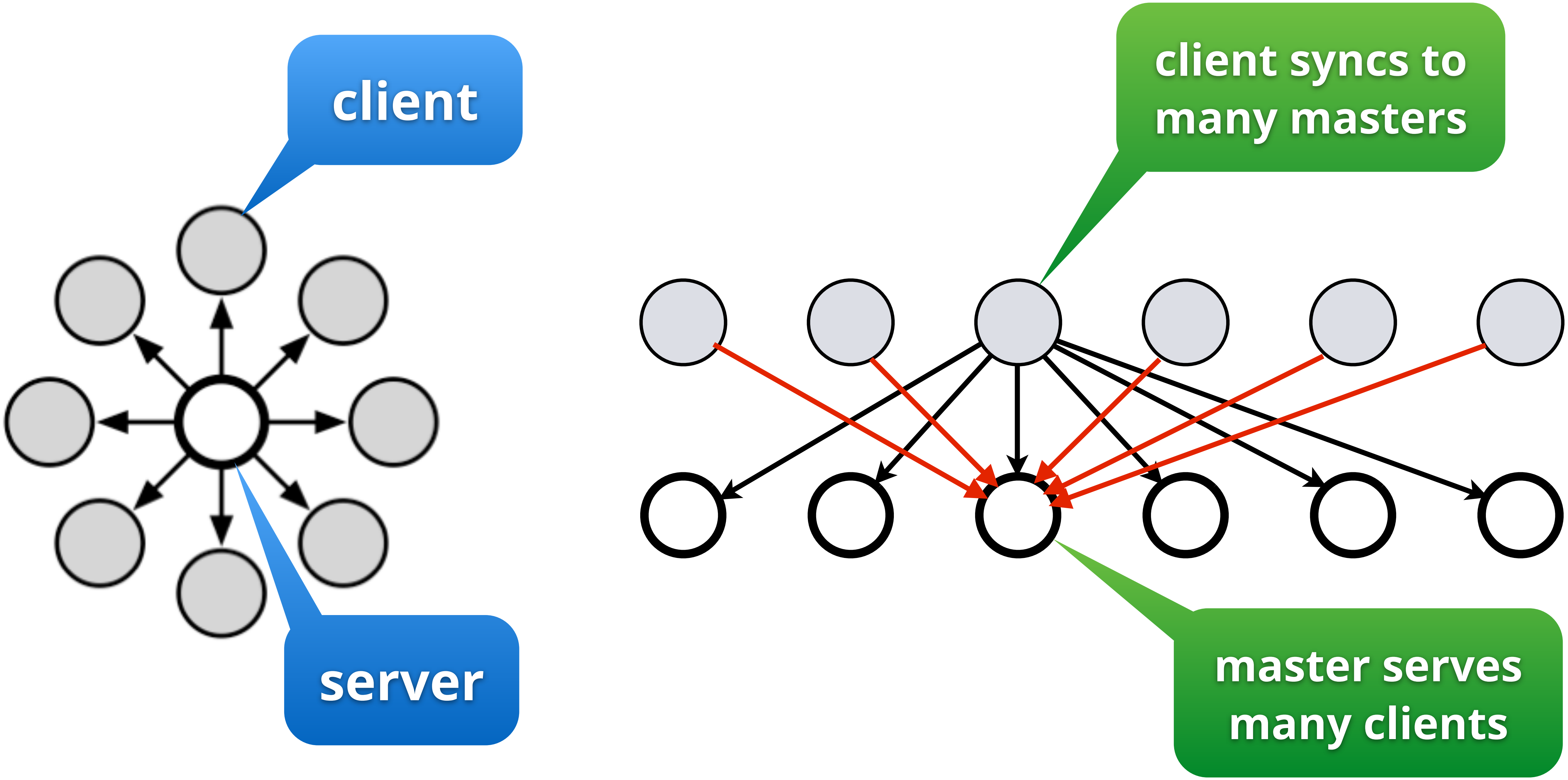
slow disks, bad memory, misconfigured machines, flaky machines, etc.

Outline

- **Motivation**
  Models, hardware
- **Bipartite design**
  Communication, key layout, recovery
- **Efficiency**
  Filters, consistency models
- **Improving the Layout**
  Submodular load balancing
- **Experiments**


P A R A M E T E R
S E R V E R

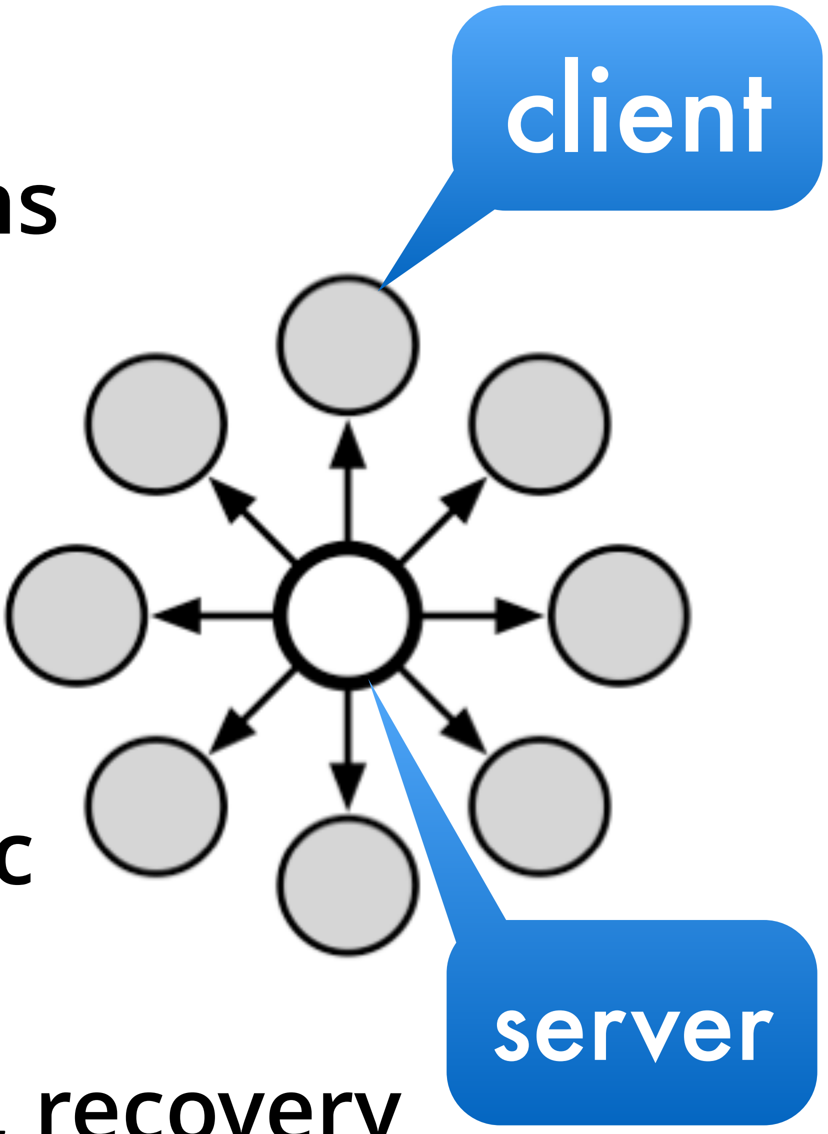
Communication pattern



put(keys,values,clock), get(keys,values,clock)

General parallel algorithm template

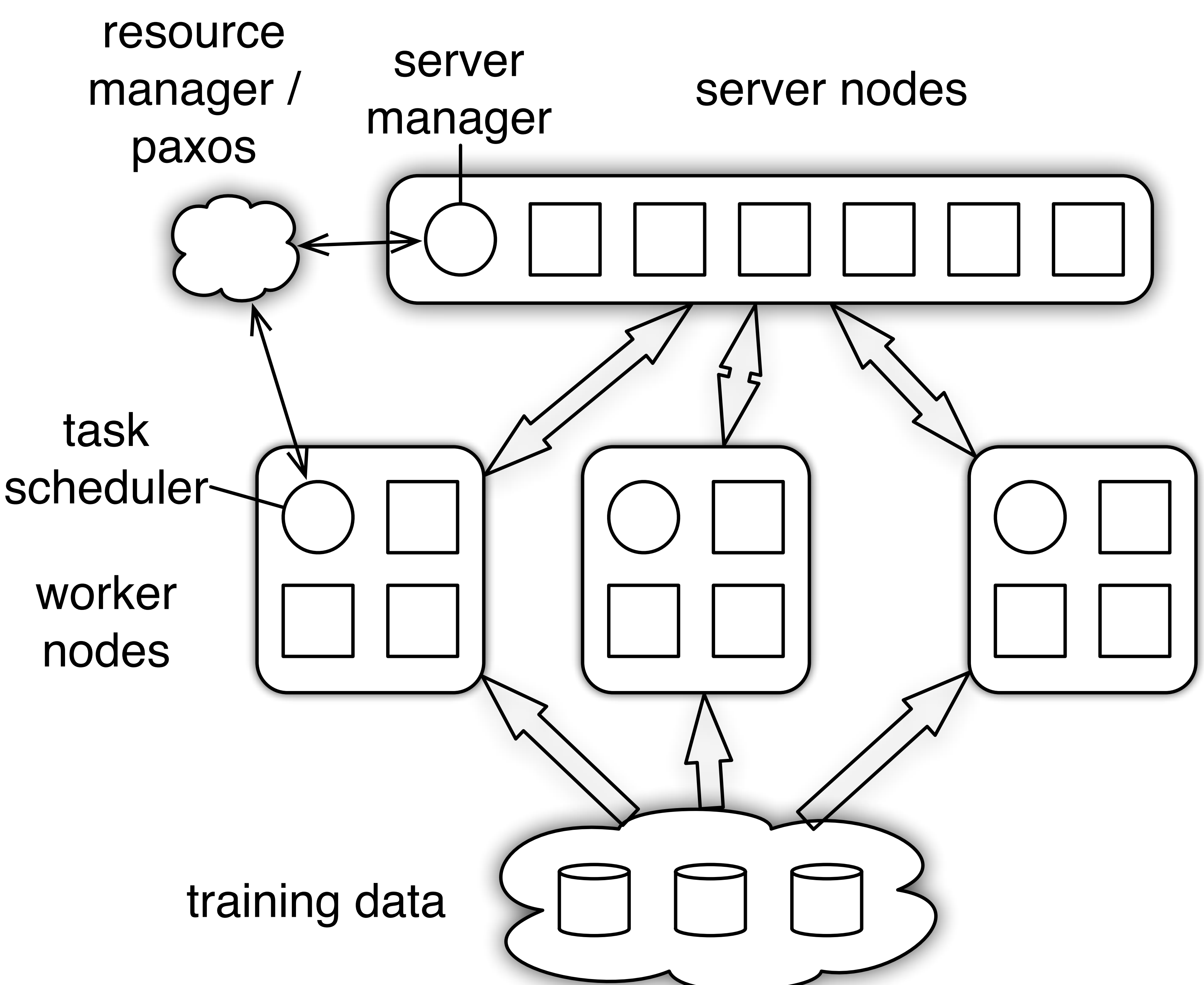
- Clients have local view of parameters
- P2P is infeasible since $O(n^2)$ connections
- Synchronize with parameter server
  - Reconciliation protocol
    average parameters, lock variables
  - Synchronization schedule
    asynchronous, synchronous, episodic
  - Load distribution algorithm
    uniform distribution, fault tolerance, recovery



Smola & Narayanamurthy, 2010, VLDB
Gonzalez et al., 2012, WSDM
Shervashidze et al., 2013, WWW

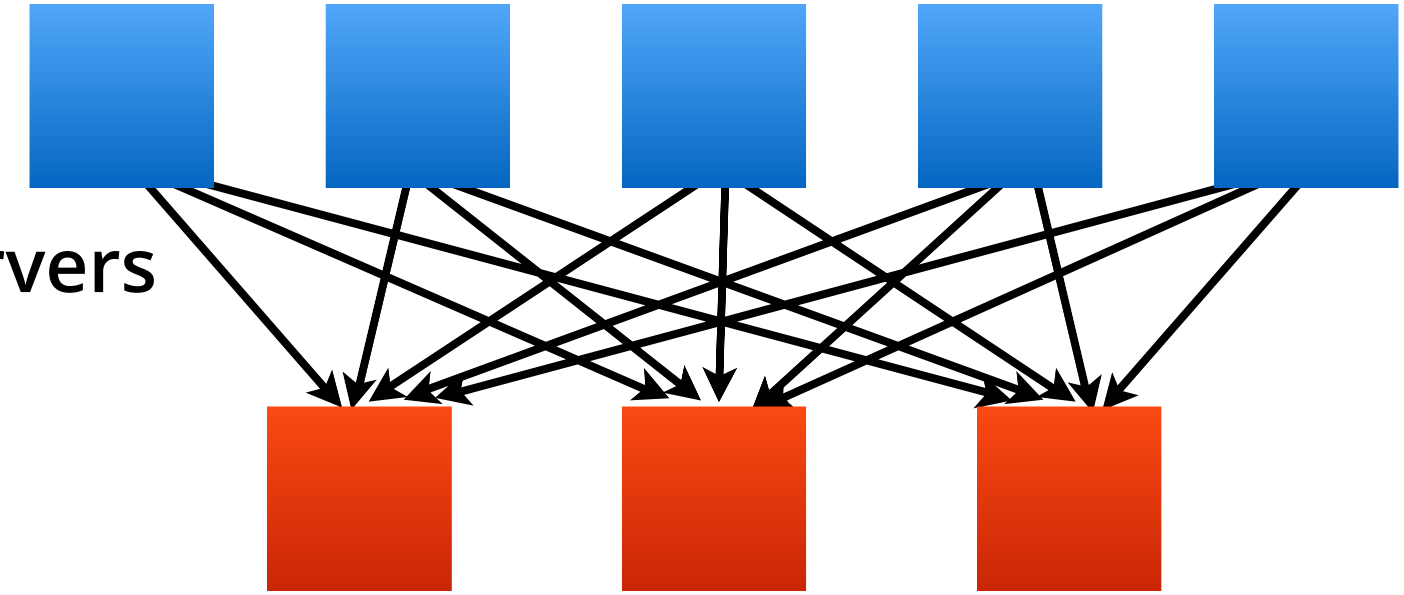
also at Google, Baidu, Facebook, Amazon, Yahoo, Microsoft, ...

Architecture




Key layout & recovery

Consistent Hashing



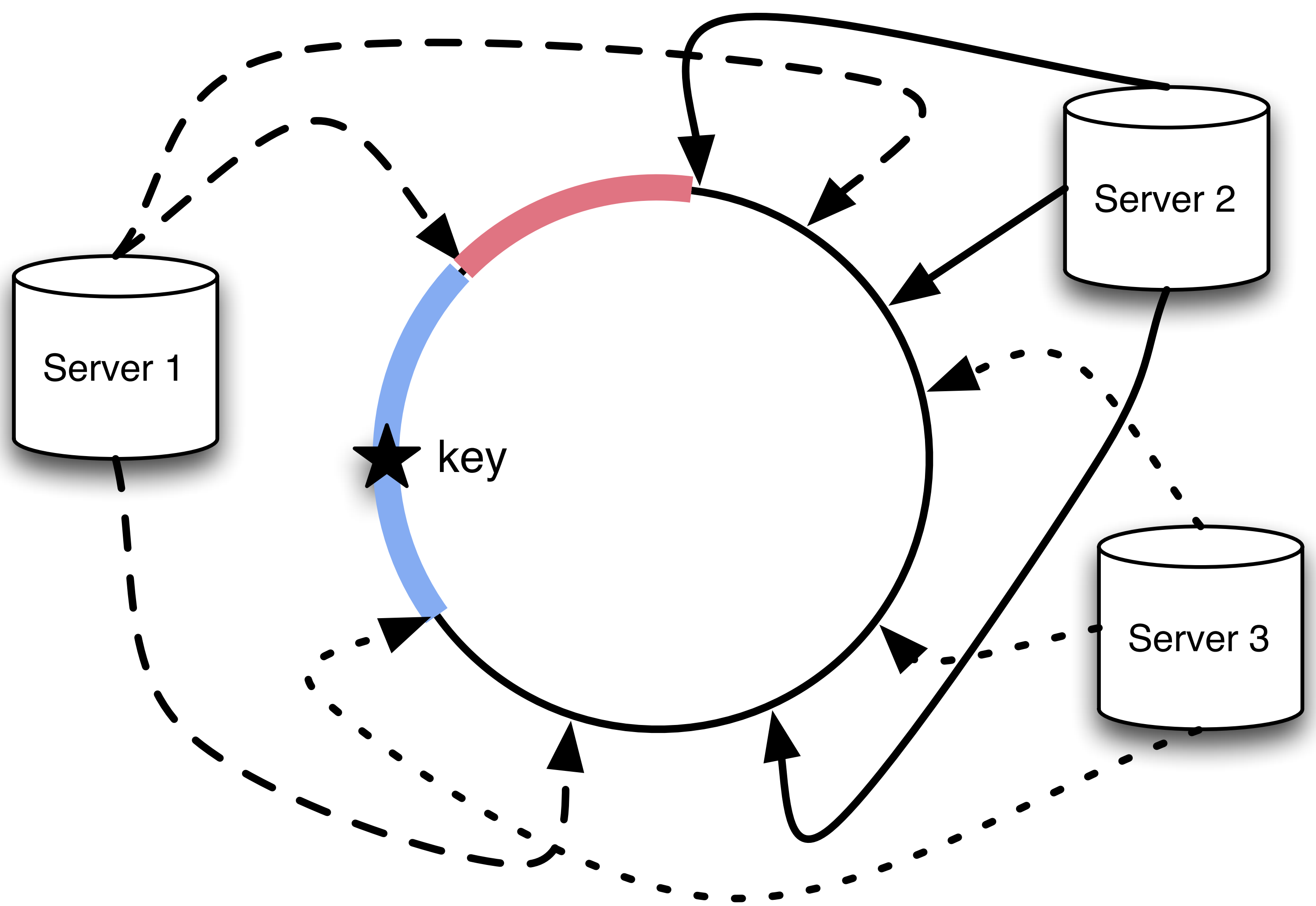
- Caching
  - Store many (key,value) pairs
  - Linear scaling in clients & servers
  - Automatic key distribution
- memcached
  - (key,value) servers
  - client access library distributes access patterns
  - randomized O(n) bandwidth
  - aggregate O(n) bandwidth
  - load balancing via hashing
  - no versioned writes / vector clocks
  - very expensive to iterate over all keys for a given server

$$m(\mathrm{key}, \mathcal{M}) = \operatorname*{argmin}_{m' \in \mathcal{M}} h(\mathrm{key}, m')$$

Google

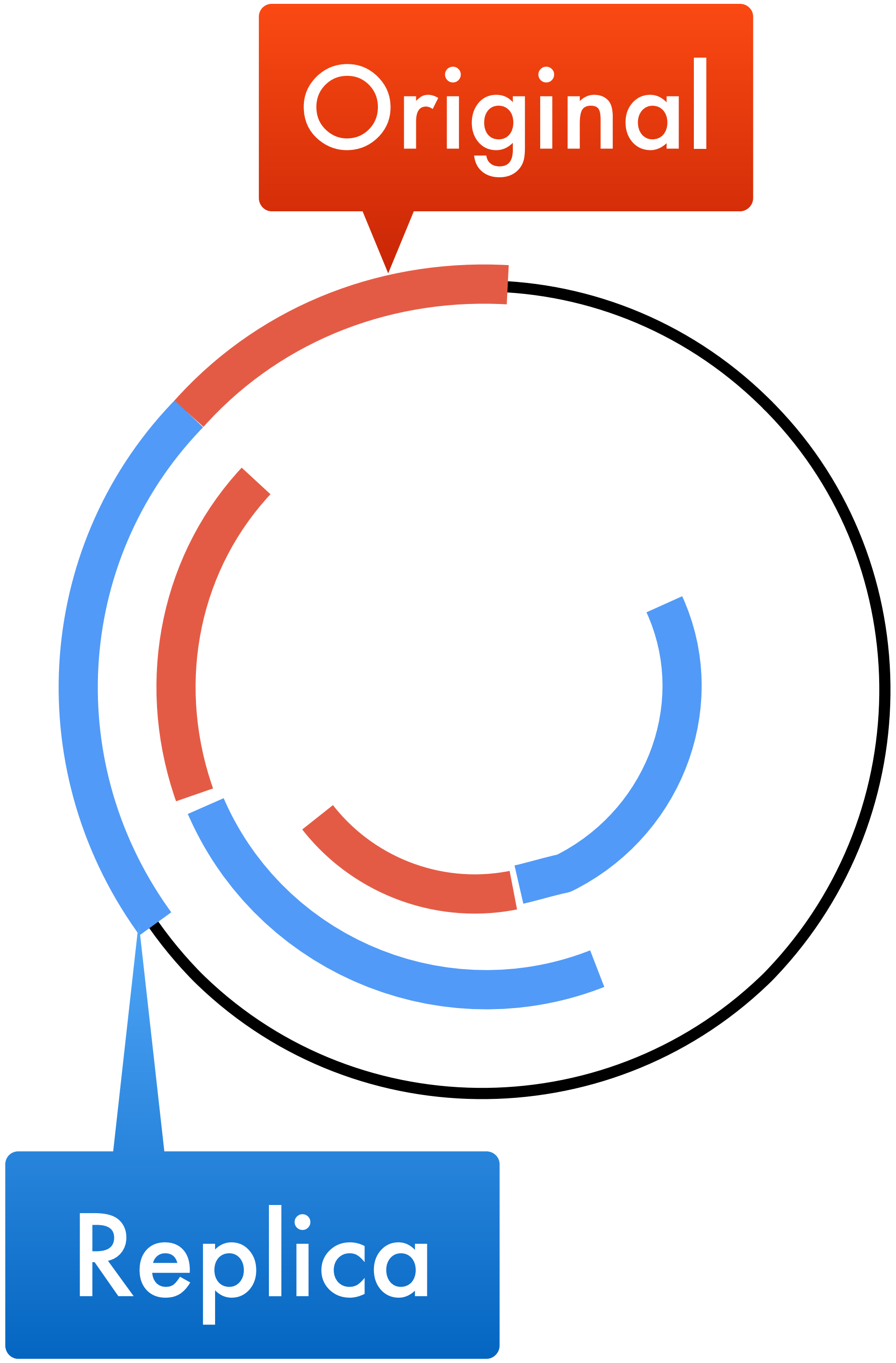
Carnegie Mellon University

Keys arranged in a DHT



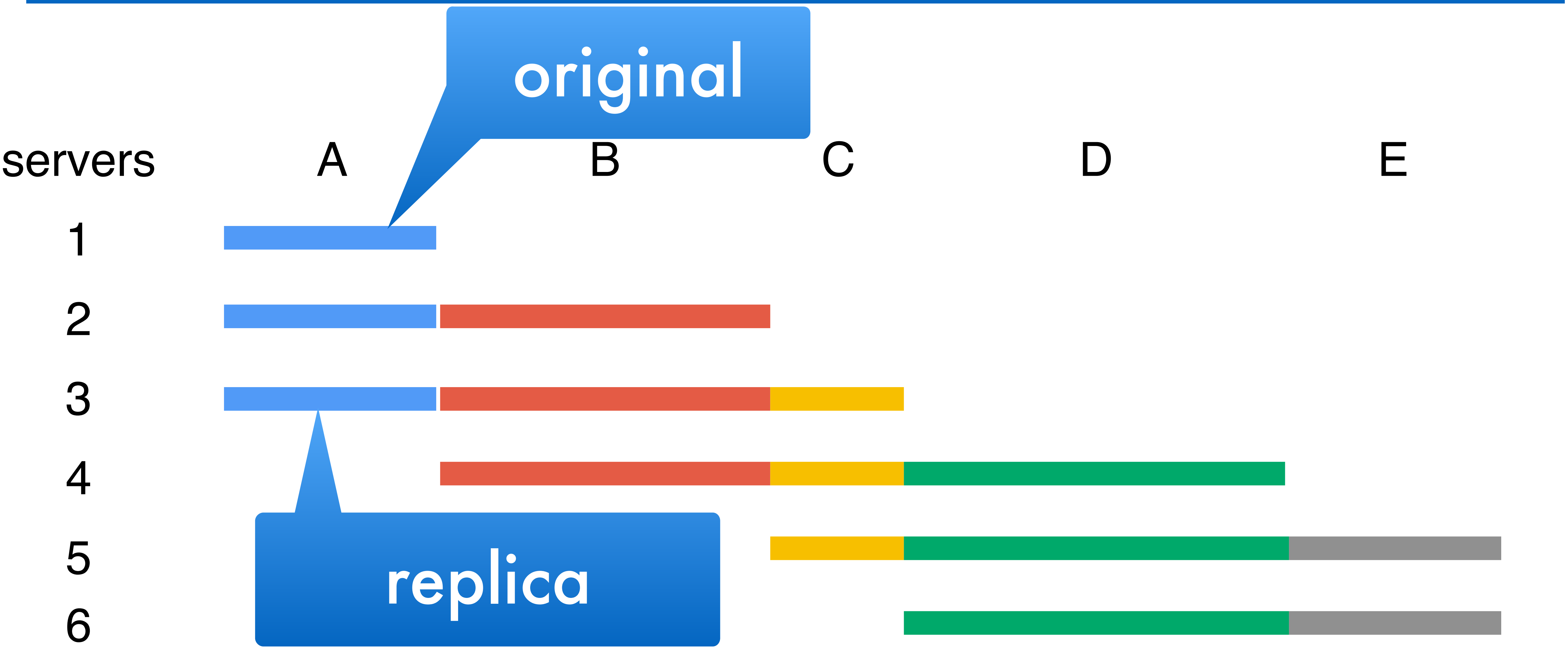
- Virtual servers
  - loadbalancing
  - multithreading
- DHT
  - contiguous key range for clients
  - easy bulk sync
  - easy insertion of servers
- Replication
  - Machines hold replicas
  - Easy fallback
  - Easy insertion / repair

Key Replication



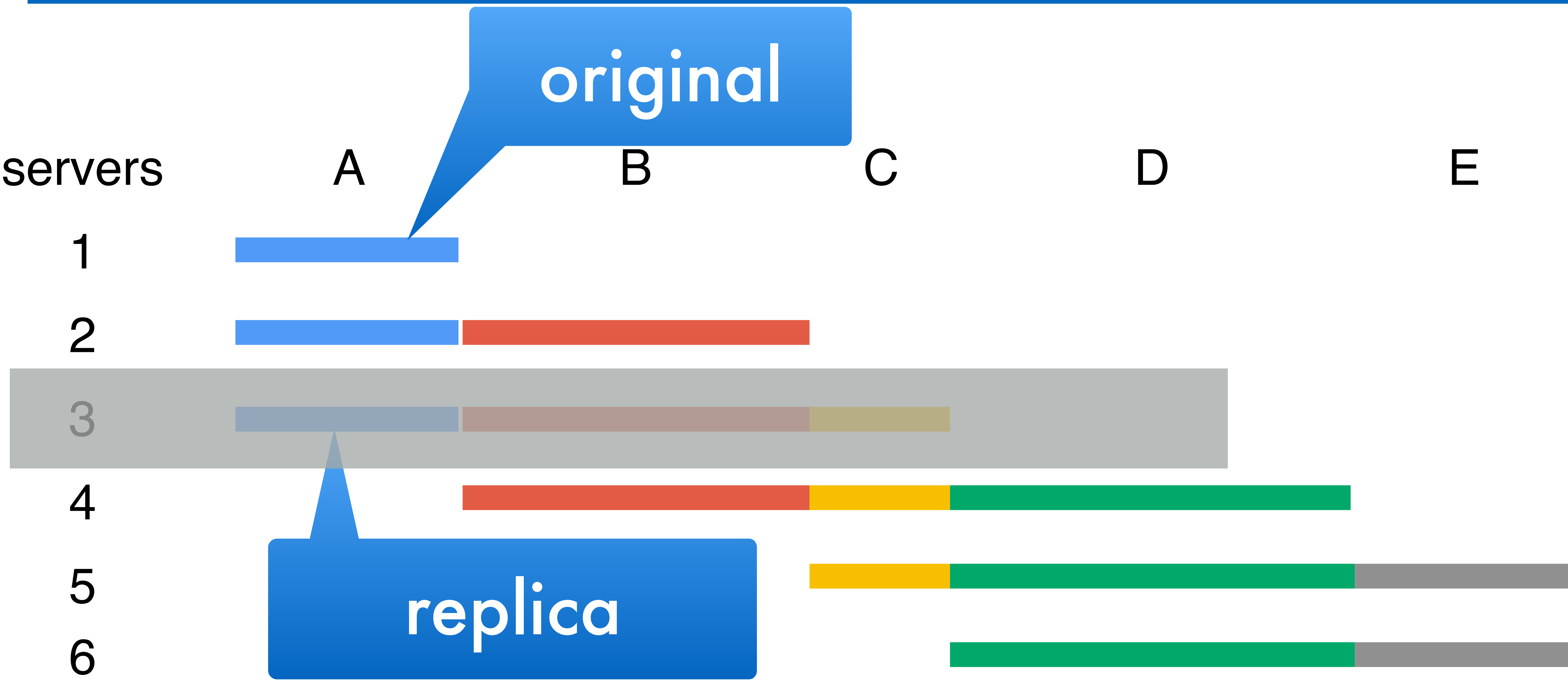
- Each segment is owned by one virtual server
- Subsequent machines hold replicas
- Easy fallback
- Easy insertion / repair
- Dynamic load balancing

Key layout

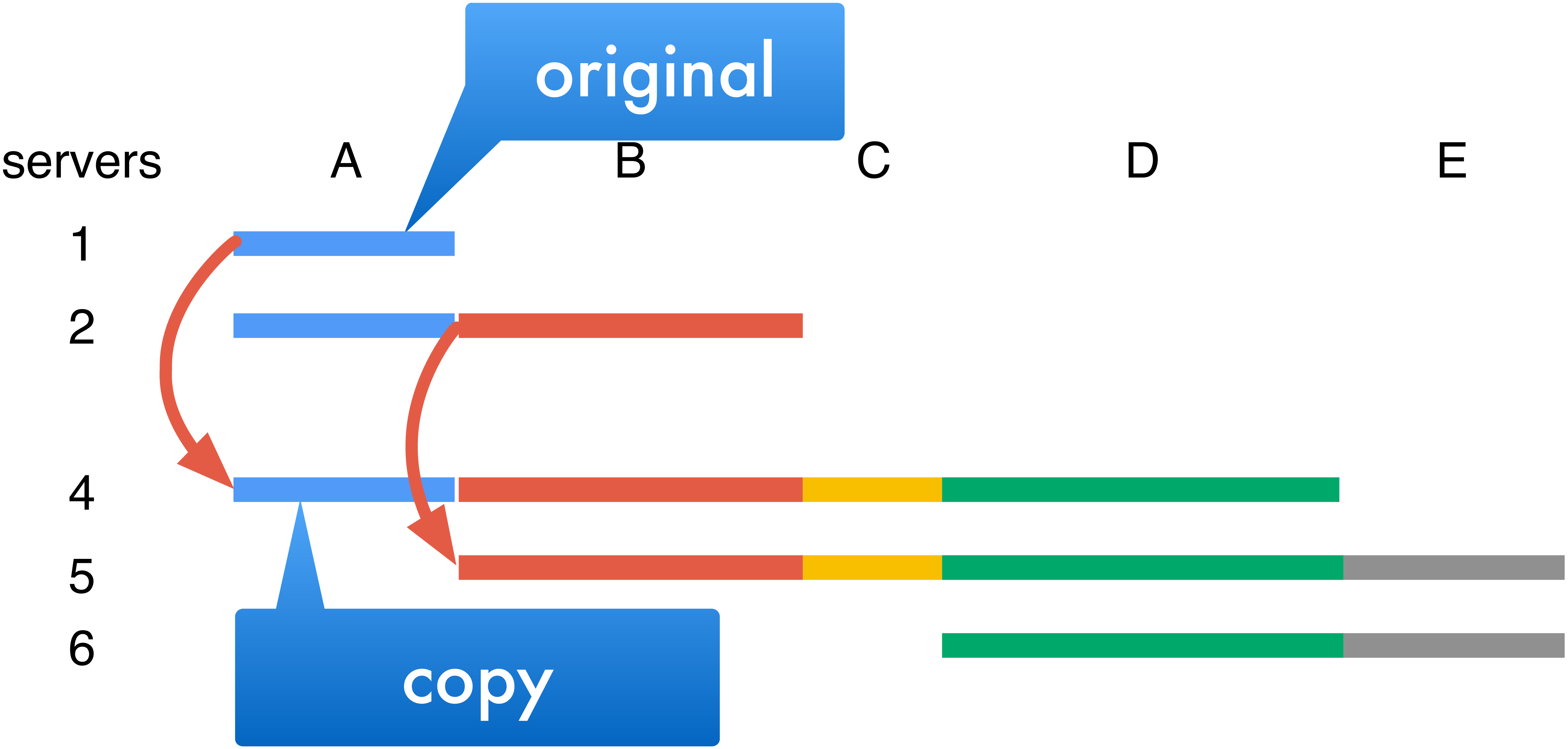


Carnegie Mellon University

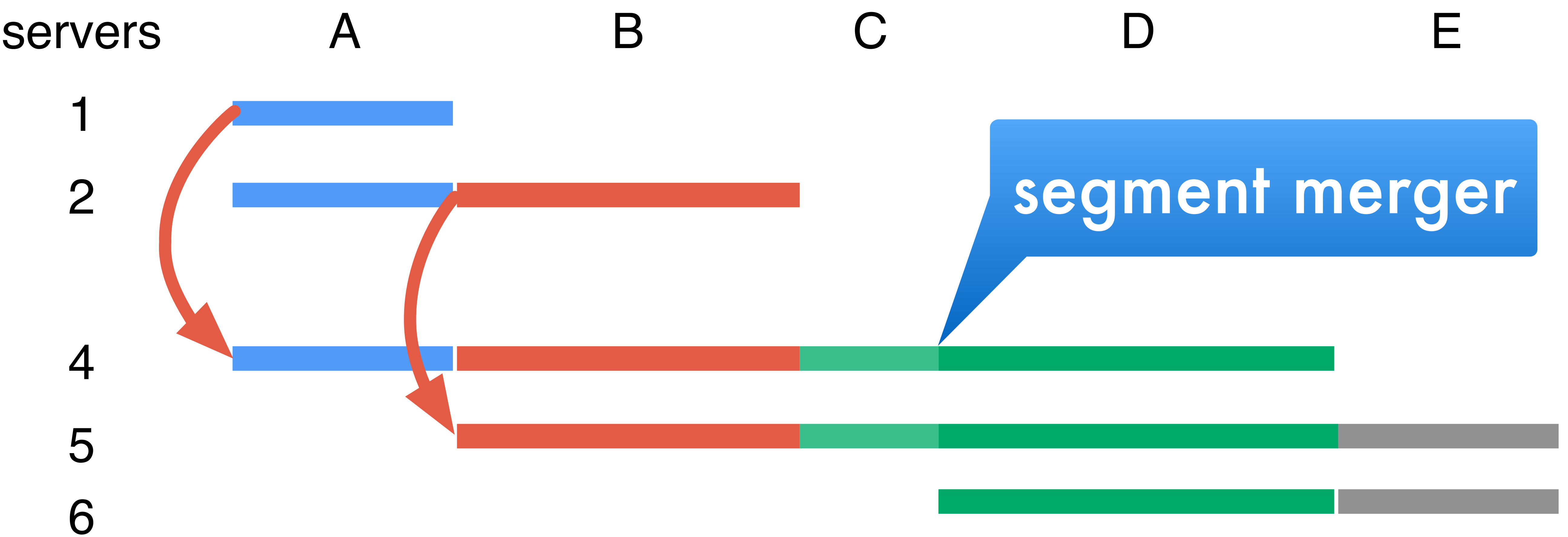
Key layout



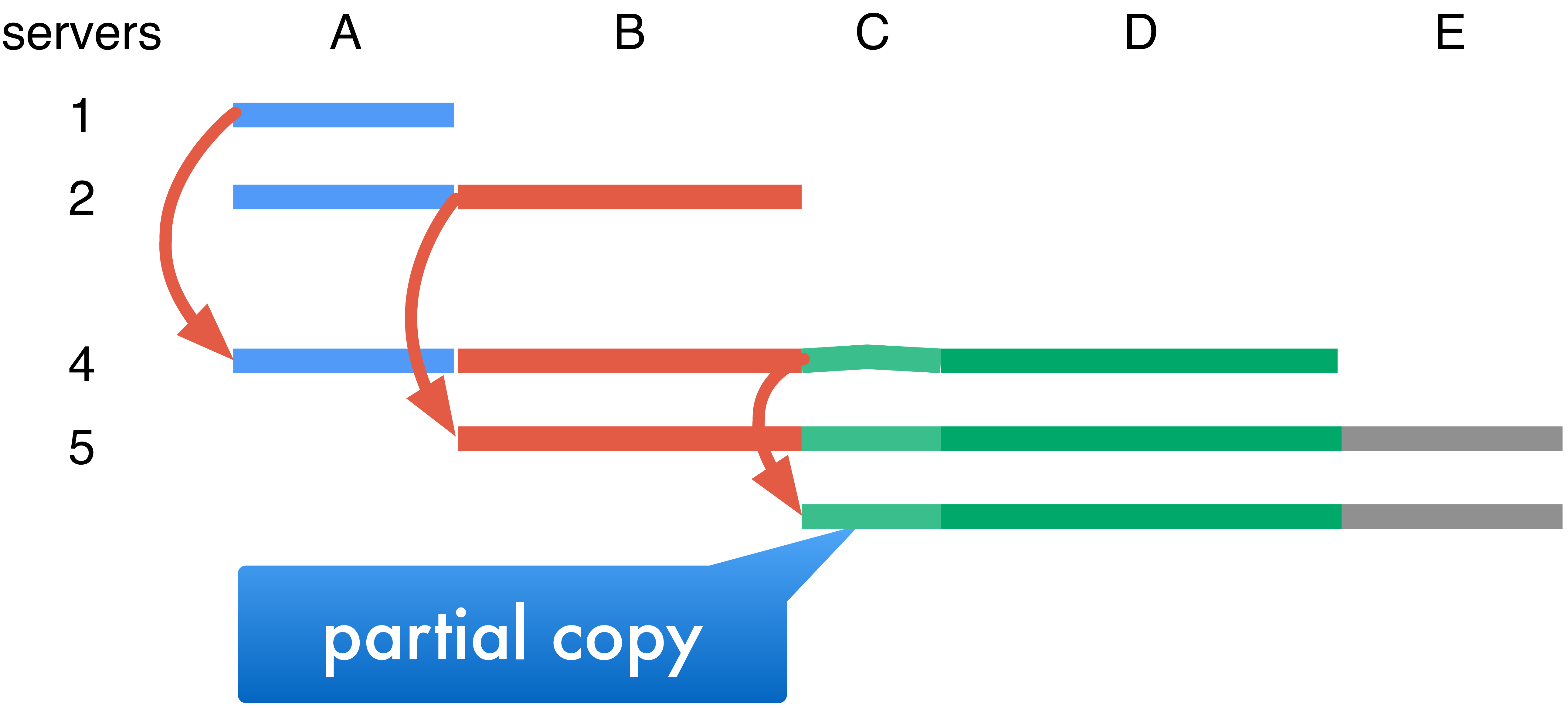
Key layout



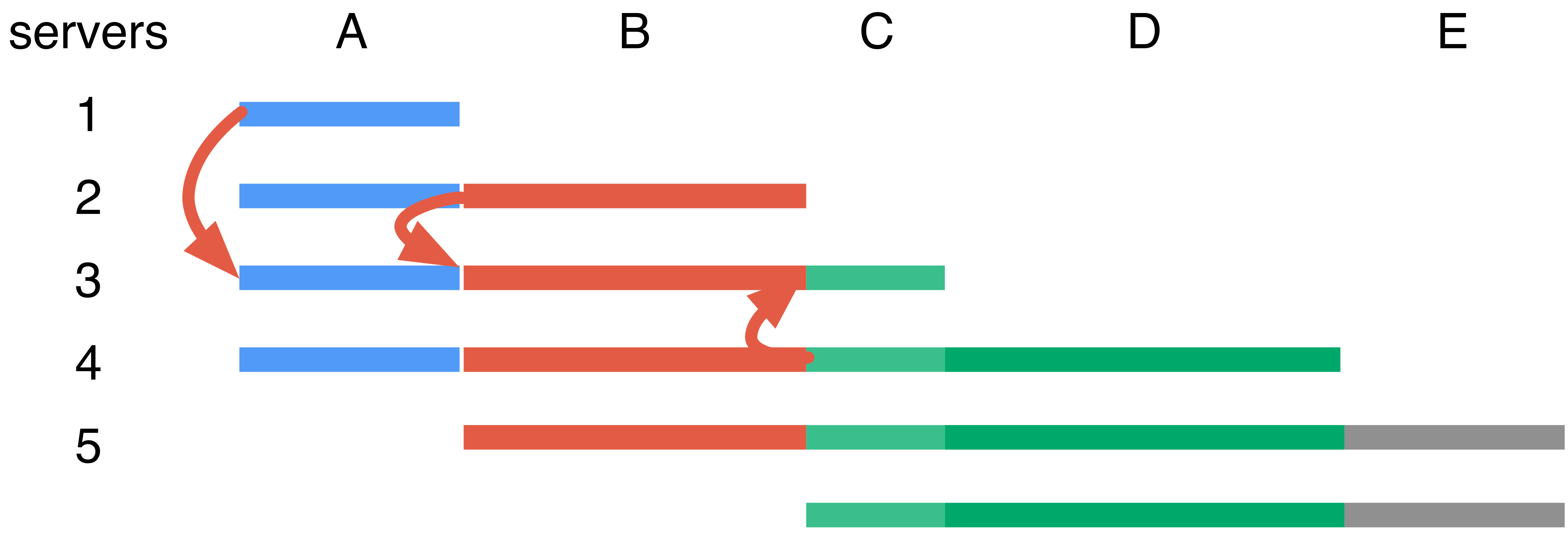
Key layout



Key layout



Recovery / server insertion



- Precopy server content to new candidate (3)
- After precopy ended, send log
- For k virtual servers this causes $O(k^{-2})$ delay
- Consistency using vector clocks

Communication

Simple API

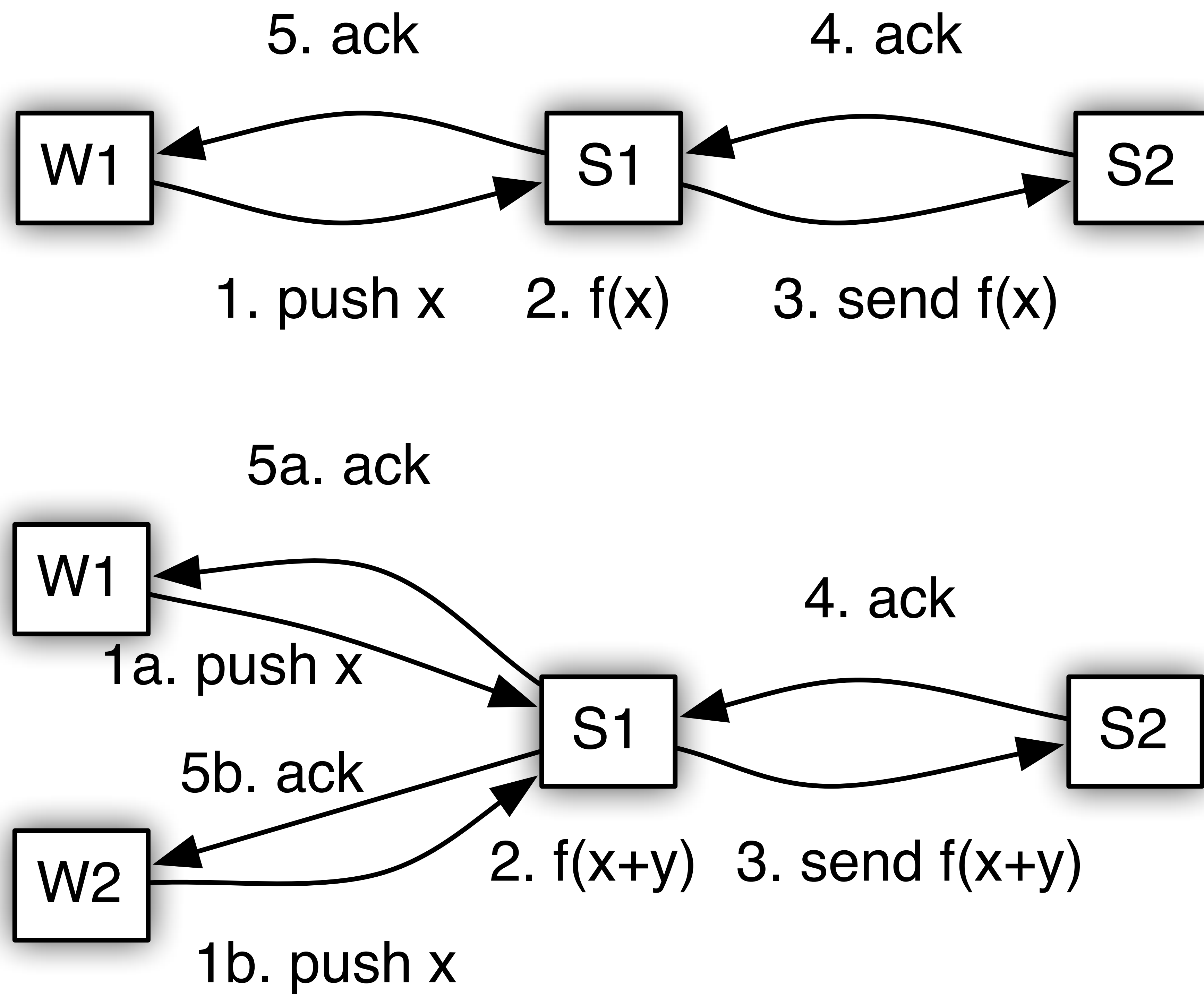
- Clients and Servers share much code
- Send data to server
  asynchronously in an interval
  `push(key_list,value_list,flag)`
- Receive data from server in an interval
  `pull(key_list,value_list,flag)`
- Avoid sending single items
  - Serialization overhead - protobuf message
  - Consistency overhead - O(c) vector clocks

Google Carnegie Mellon University

Batched Communication

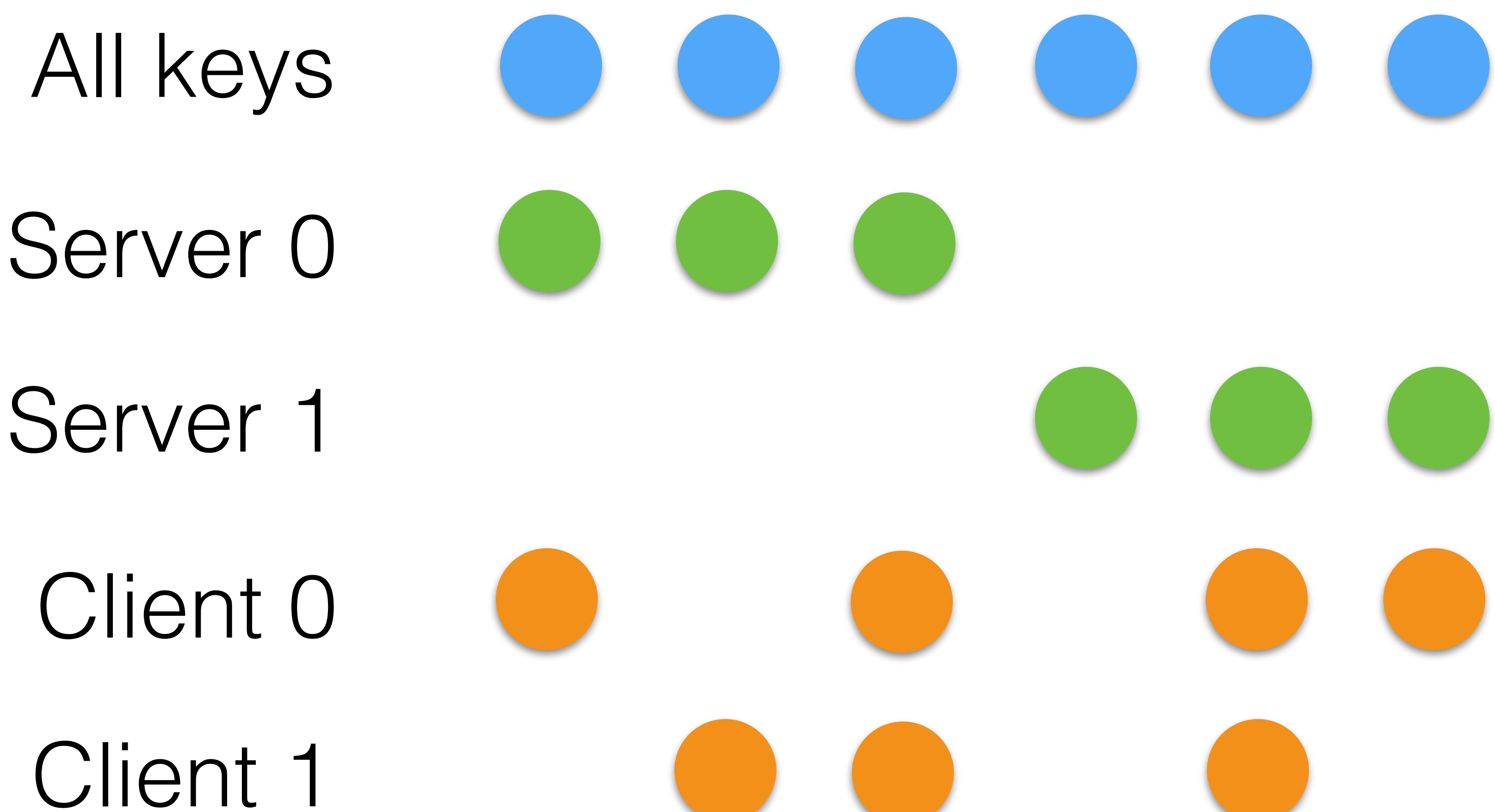
- Overhead of sending individual K/V is large
  - 10^{10}—10^{14} packages
  - Package header (e.g. TCP/IP) matters
  - Horrible examples: memcached, YahooLDA (yes, it's easy to beat YahooLDA ...)
- Communicate only when
  - Finish one local "iteration" (processed a group of samples or features)
  - Reached the end of a specific time window (prevent stale data)

Message Aggregation on Server



Send as little as possible

- Only send data the receiver needs
  - A server node maintain segments of keys
  - Client nodes may have different working sets



Outline

- **Motivation**
  Models, hardware
- **Bipartite design**
  Communication, key layout, recovery
- **Efficiency**
  Filters, consistency models
- **Improving the Layout**
  Submodular load balancing
- **Experiments**


Filters

Message Compression

- Convergence speed depends on communication efficiency
  - Sending (key,value) pairs is inefficient
    Send only values (cache key list) instead
  - Sending small gradients is inefficient
    Send only sufficiently large ones instead
  - Updating near-optimal values is inefficient
    Send only large violators of KKT conditions
- Filter data before sending

Key compression

- Data Compression
  - Google Protobuf
  - Zippy
- Ignore keys if possible
  client 0 sends to server 0
  - time 1: (2,2.3), (4,6.1), (8,9.9)
    . . .
    time 6: (2,5,4), (4,2.5), (8,2.9)

Both sender and receiver cache the key list. If hit cache, then send checksum only

Quantization Filter

- Gradient from each client requires 16 bytes each (gradient / preconditioner)
- Precision is often not required
  - Reduce bit resolution (double -> float)
  - Quantize even further (8 bit often enough)
- Randomized rounding

$$g_{\mathrm{rr}} = \left\lfloor \frac{g - g_0}{\epsilon} \right\rfloor + \xi \text{ where } \xi \sim \mathrm{Bin}\left(\frac{g - g_0}{\epsilon} - \left\lfloor \frac{g - g_0}{\epsilon} \right\rfloor \right)$$

Google

Carnegie Mellon University

Sparsification

Eliminate entire coefficients

- **Constant probability**

$$g_{\text{sparse}} = \pi^{-1}\xi g \text{ where } \xi \sim \text{Bin}(\pi)$$

- **Duffield-Lund-Thorup sampling**
  - **Each coordinate gets priority**

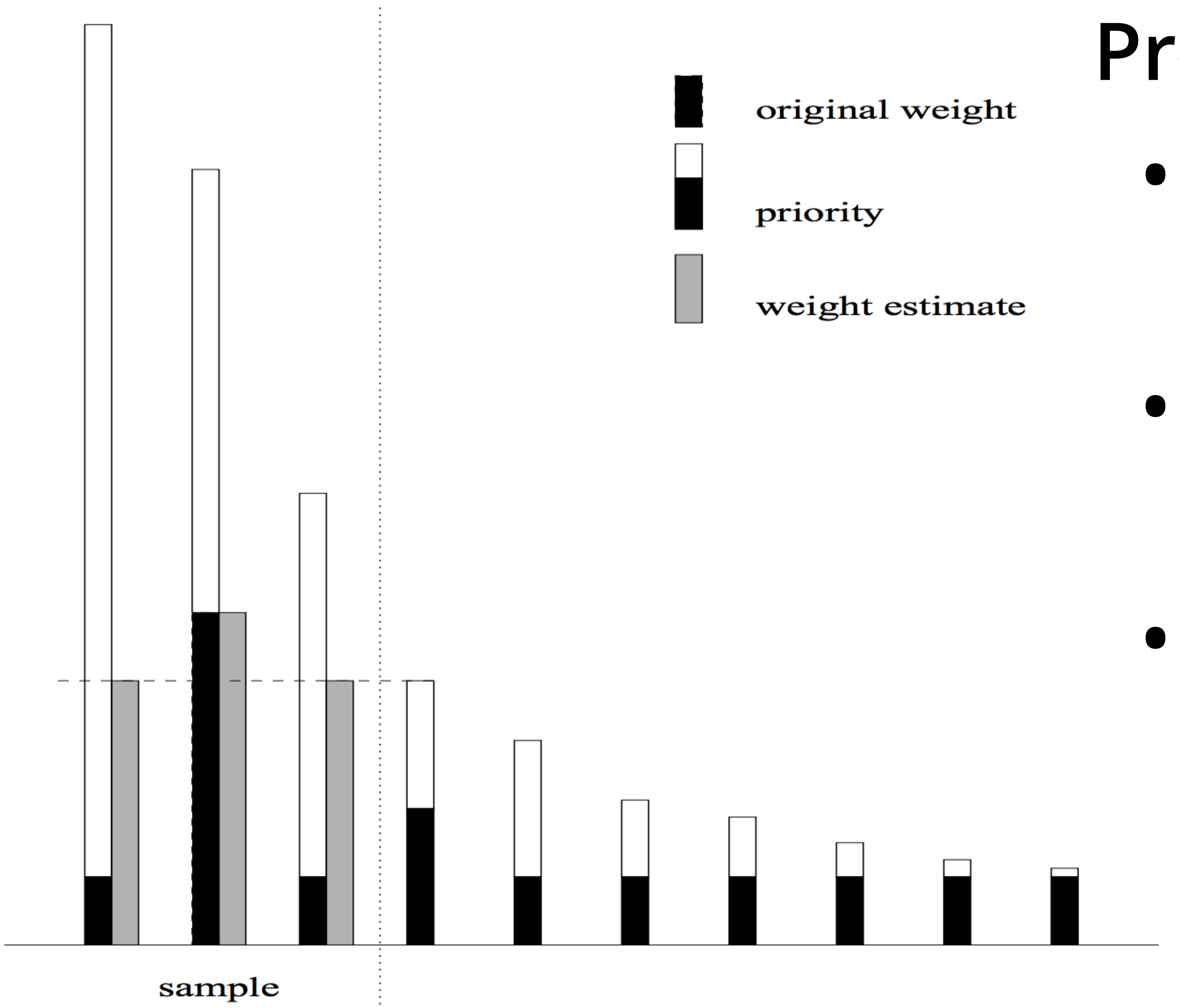
$$q_i = \frac{|g_i|}{\alpha_i} \text{ where } \alpha_i \sim U[0,1]$$

  - **Pick top k terms and weigh with**

$$\max(|g_i|, |g_{k+1}|/\alpha_{k+1})$$

Sparsification

Priority sampling for estimation of arbitrary subset sums



Proof

- Fix all weights but one, say i
- We have threshold t
- Probability that above threshold

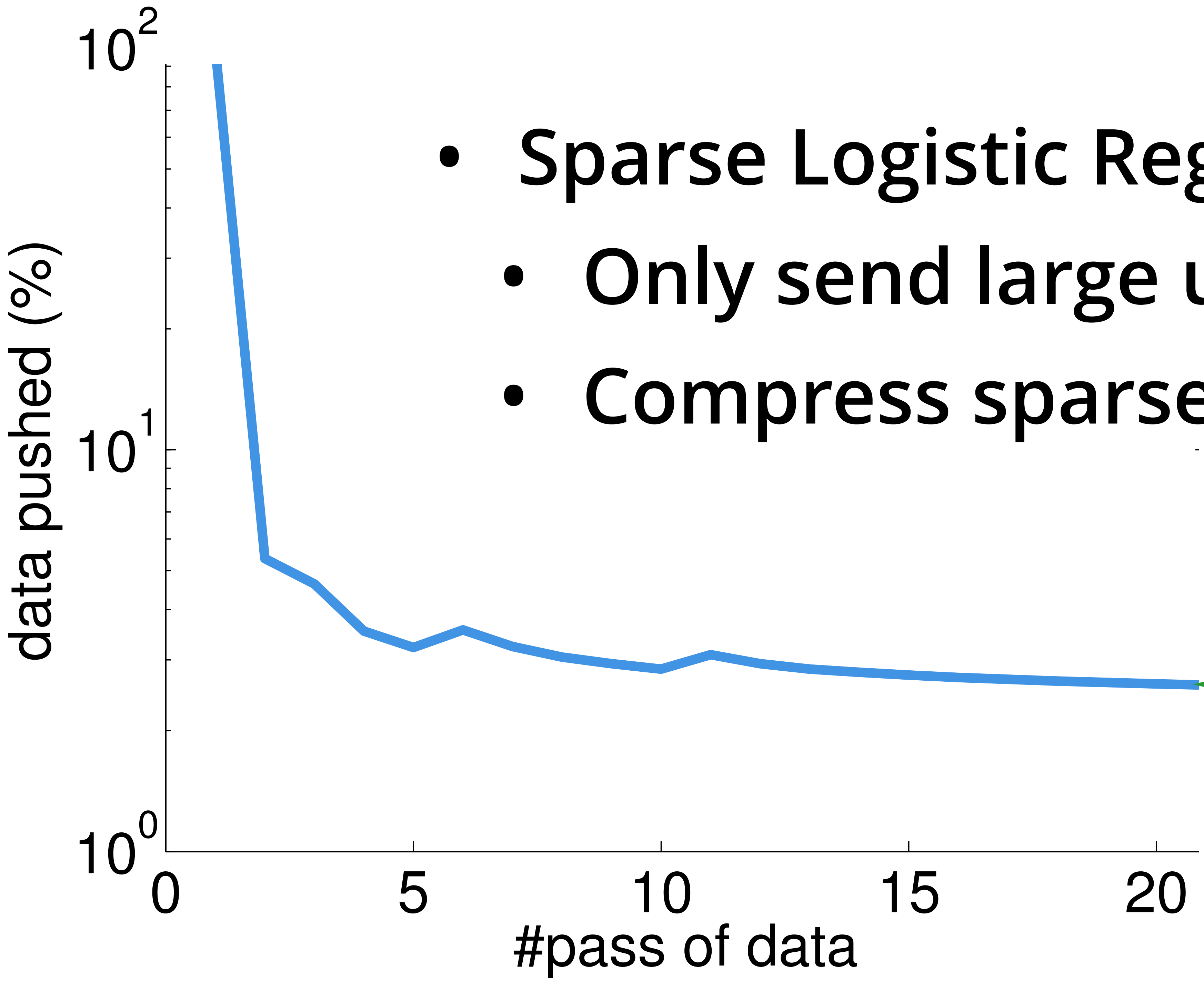
$$\min(1, |g_i|/\tau)$$

More Filters

- Scheduling
  have controller decide when to send
  (this requires very smart controller - difficult)
- Filtering (easier)
  have algorithm decide when to shut up
  - Gradient (only send large gradients)
  - KKT (only send variables violating KKT)

Filters in practice

- Sparse Logistic Regression
  - Only send large updates
  - Compress sparse value list



20x compression

Clocks and Consistency

Consistency Zoo

- Samplers only need loose synchronization
(large delay, eventual consistency)
- Hogwild
(fully asynchronous, unclear how messy)
- Distributed proximal gradient
(needs bounded delay, but delay differs)
- Brittle ML algorithms (off the shelf)
(fully synchronous, no delay)

Consistency Zoo

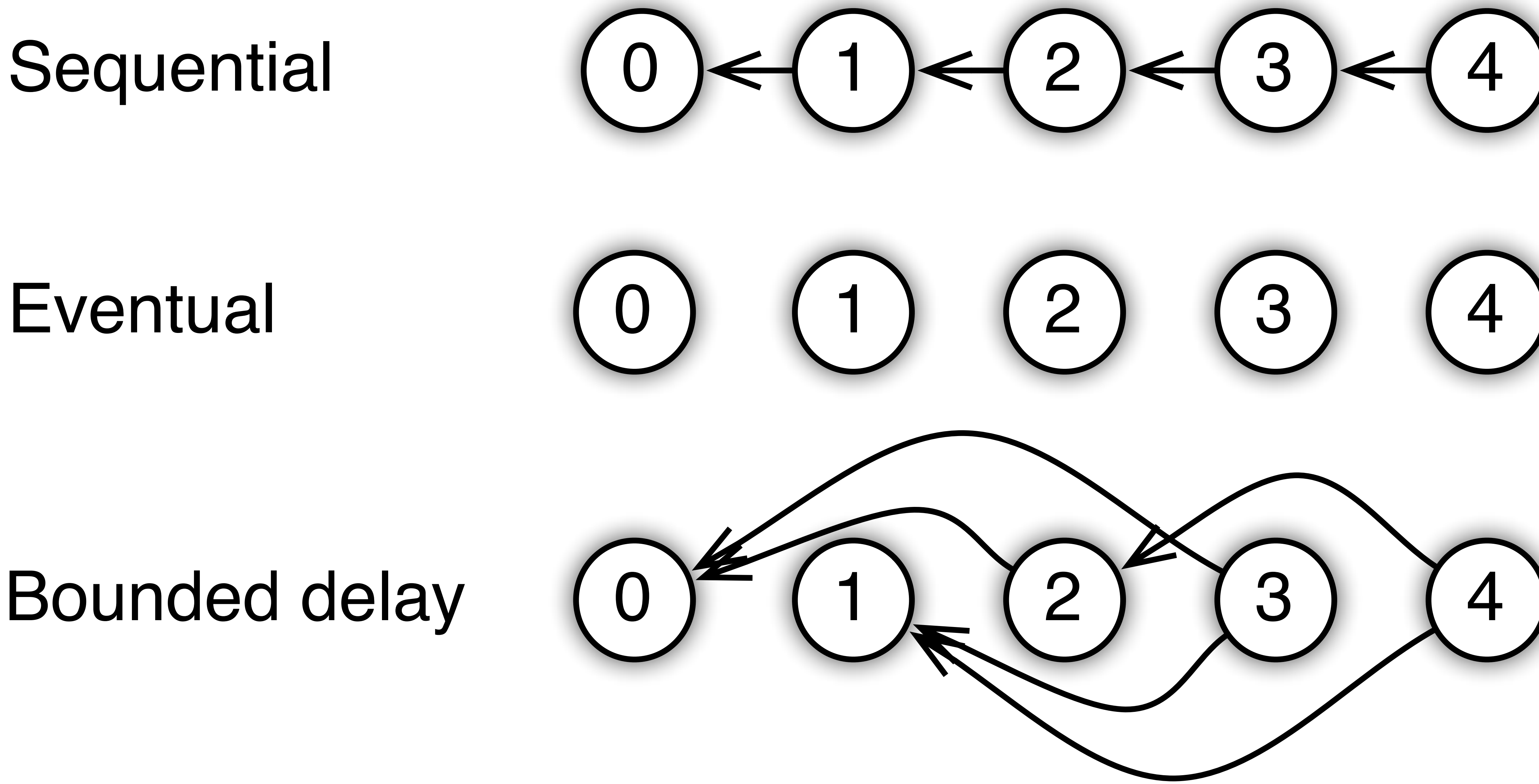
- Samplers only need loose synchronization
(large delay, eventual consistency)
- Hogwild
(fully asynchronous, unclear how messy)
- Distributed proximal gradient
(needs bounded delay, but delay differs)
- Brittle ML algorithms (off the shelf)
(fully synchronous, no delay)

Which side do you pick?

Google

ie Mellon University

Consistency models

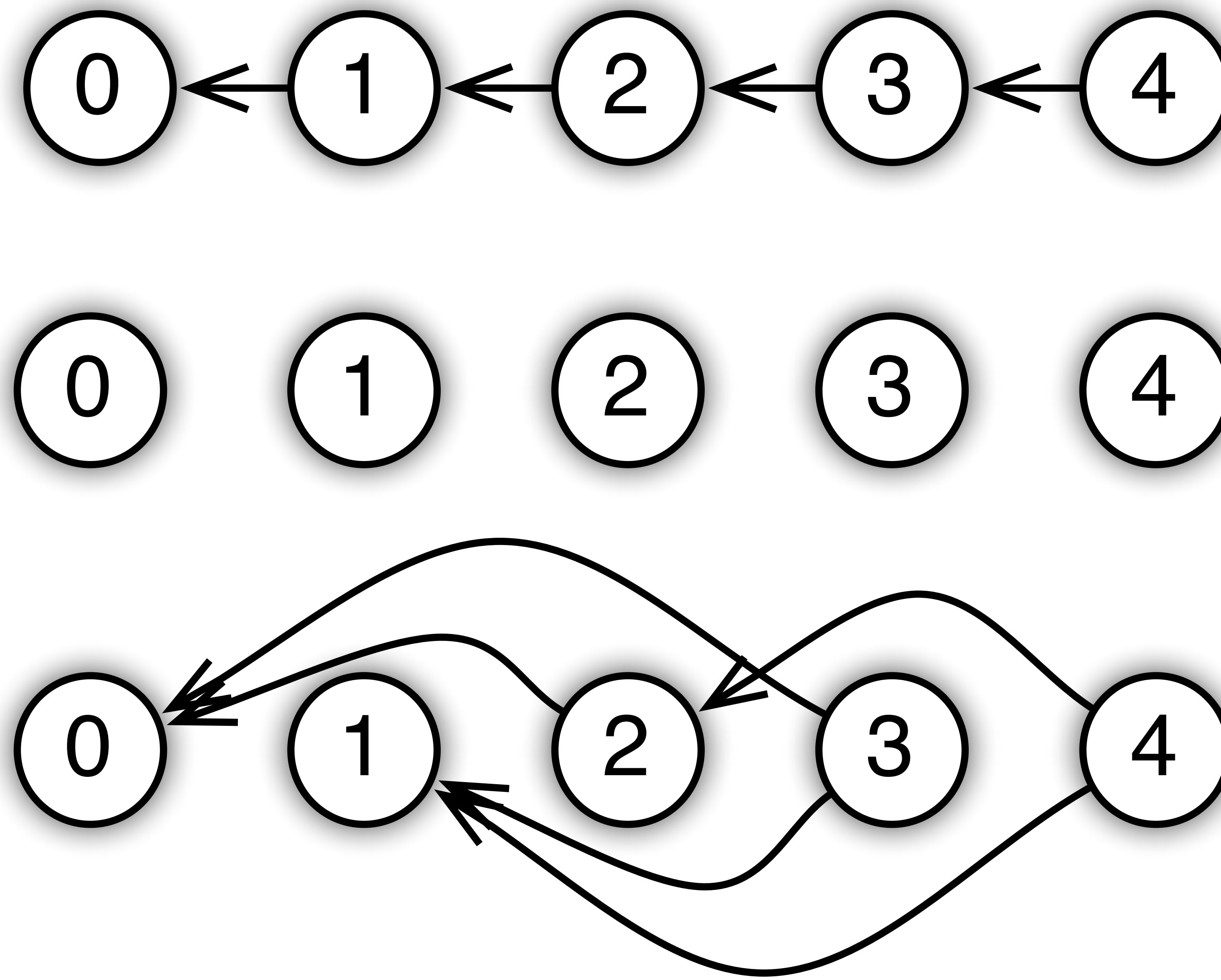


Consistency models

Sequential

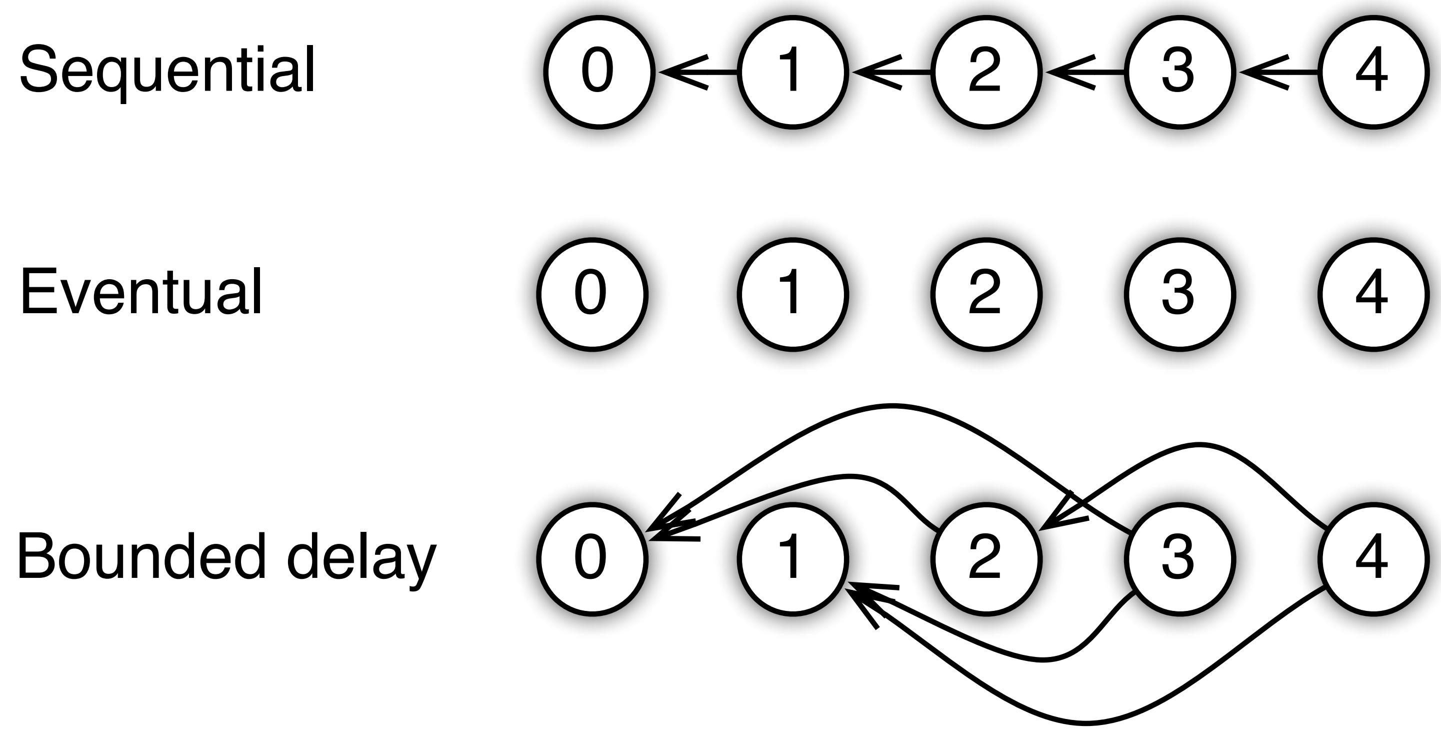
Eventual

Bounded delay



via task processing engine on client/controller

Consistency models



- Change dependency on the fly
- Task granularity programmatically defined (small or large tasks)
- Subtree controlled by worker

Vector Clocks for Ranges

- Keep track of when we received an update from a client / server.
- For c clients this means O(c) metadata
  This is impossible to store per key (Dynamo)
- Very cheap and feasible for ranges
- When inconsistent ranges, split segments
  [A,D] splits into [A,B], [B,C] and [C,D] when receiving message for [B,C]
- This is infrequent + defragmentation

Vector Clocks for Ranges

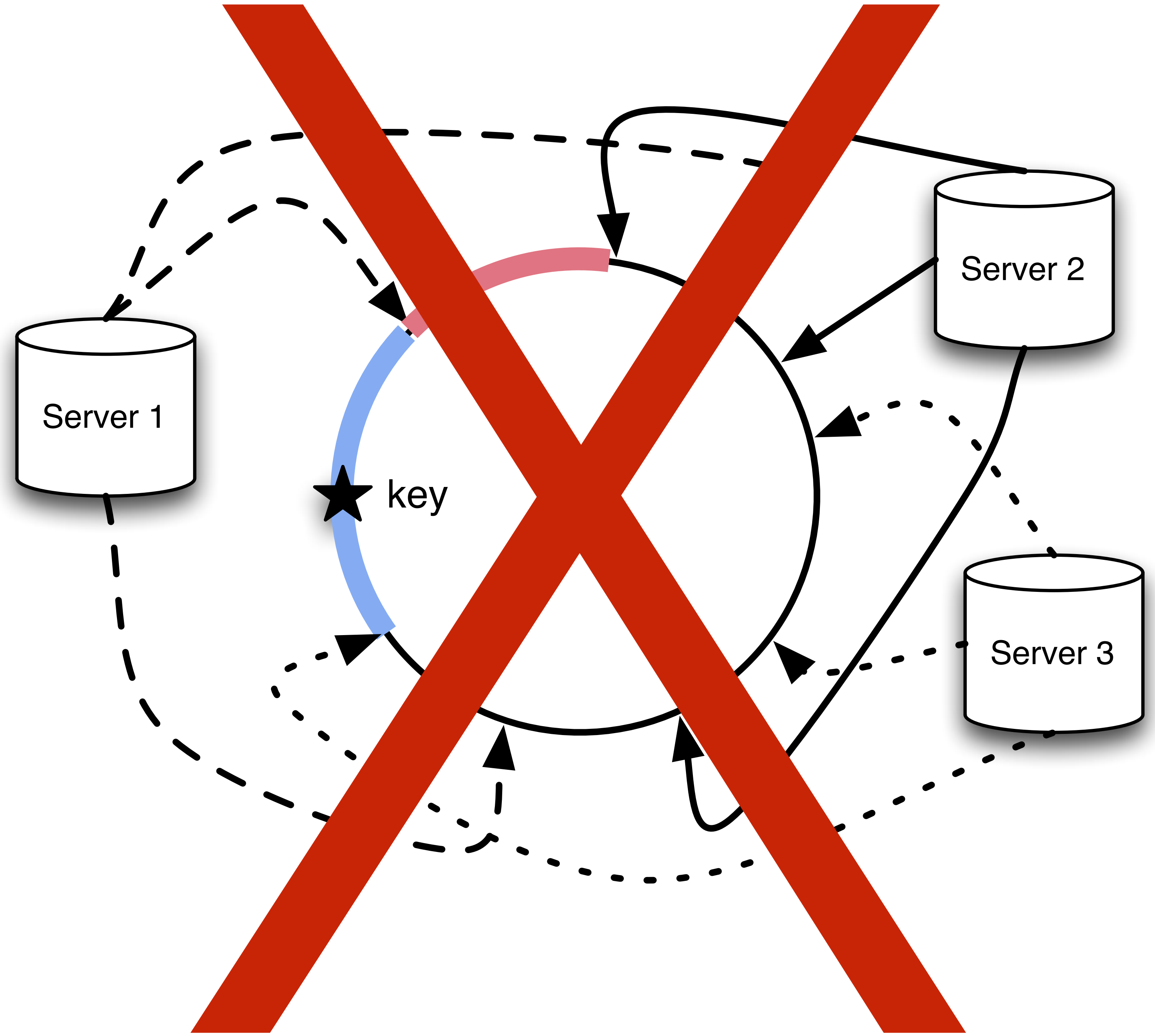
- For each (key,value) pair we know all timestamps from all clients
- If client dies and restarts, we know whether we already received the message
- Use with dependency DAG

Outline

- **Motivation**
  Models, hardware
- **Bipartite design**
  Communication, key layout, recovery
- **Efficiency**
  Filters, consistency models
- **Improving the Layout**
  Submodular load balancing
- **Experiments**

Google

Carnegie Mellon University



Improved Key Layout

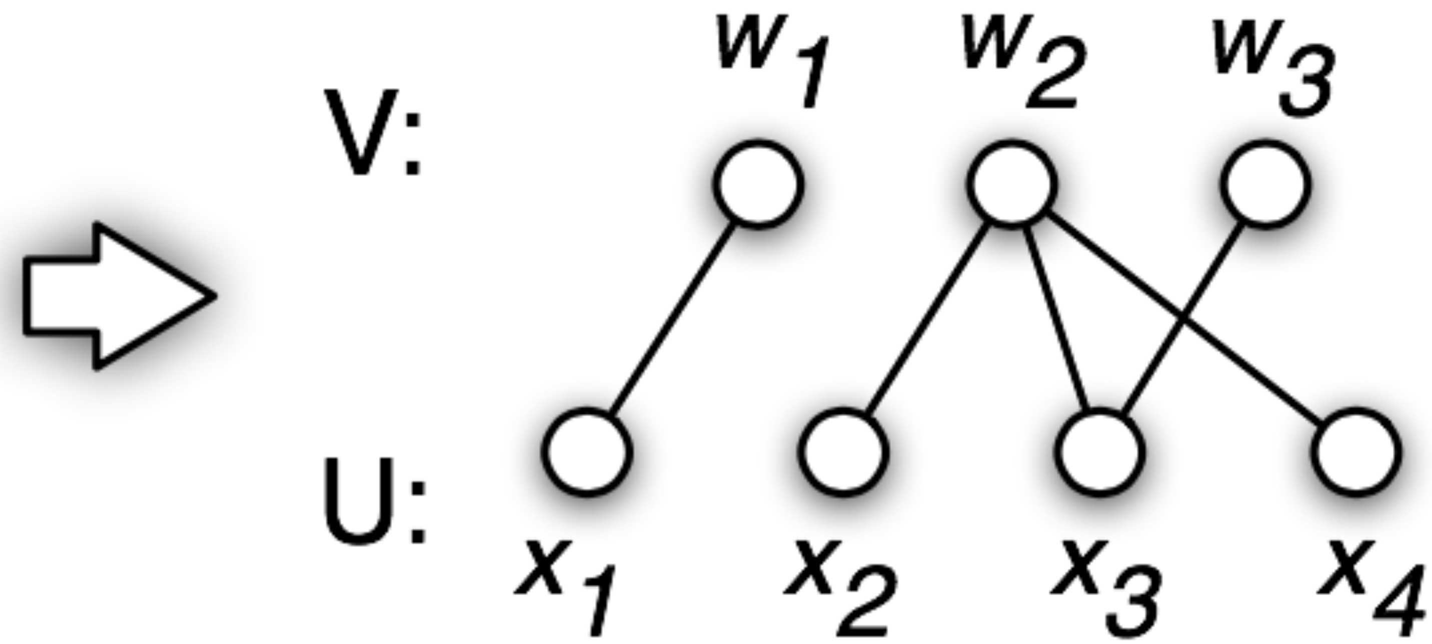
Local Key Distribution

$x_1 = (.1, \_, \_)$

$x_2 = (\_, .3, \_)$

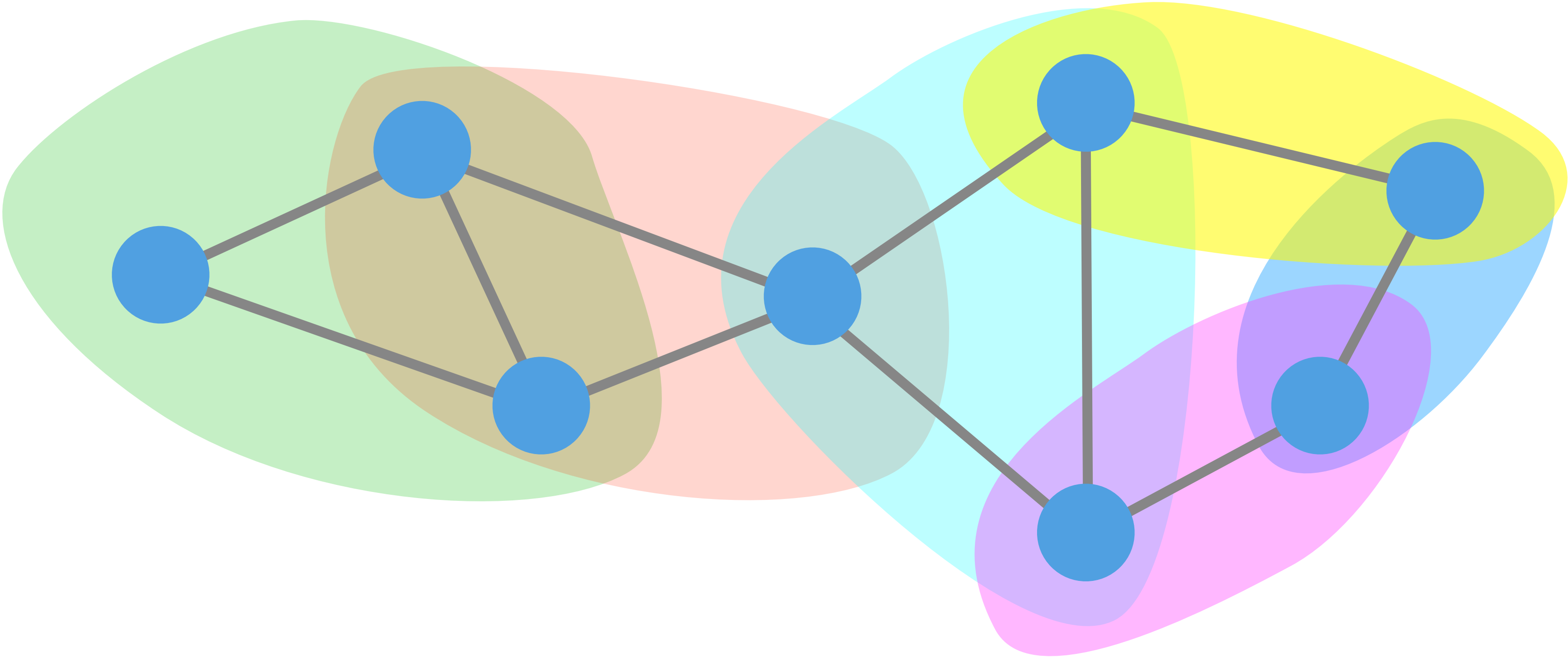
$x_3 = (\_, .4, .3)$

$x_4 = (\_, .9, \_)$



- Randomly partitioning data leads to lots of network traffic between clients & servers
  - Clients: documents, user activity (needs to cache all relevant parameters)
  - Servers: parameters

Local Key Distribution



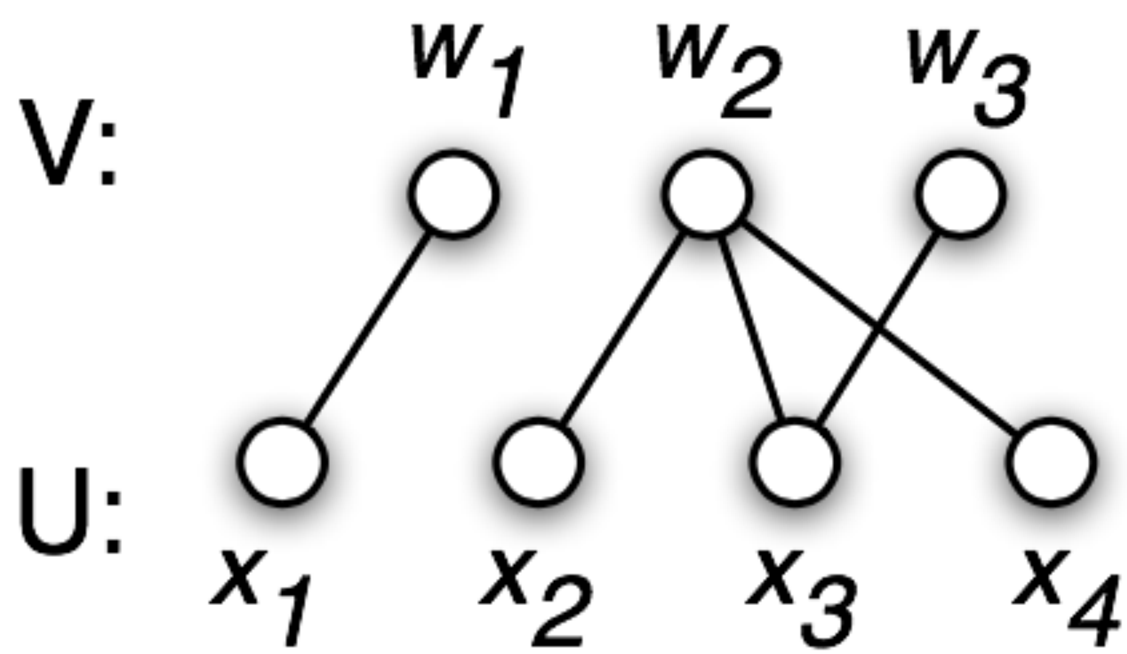
- Randomly partitioning data leads to lots of network traffic between clients & servers
  - Clients: vertices
    (needs to cache all clique potentials)
  - Servers: cliques

Goals

- Memory
  Must not exceed client memory allowance (cache all relevant variables)
- Work
  Should balance workload over clients
- Network
  Should minimize communication cost
- Without loss of generality assume bipartite graph to be partitioned

Memory

- Graph G(U,V,E)
- Select vertices in U with few neighbors
- Minimizing memory



$$\text{minimize} \quad \max_i |\mathcal{N}(U_i)| \quad \text{where } \mathcal{N}(U_i) := \bigcup_{u \in U_i} \mathcal{N}(u)$$

worst client

memory load

neighbors in V

Memory

- # Neighbors of U_i is a submodular function (if v already a neighbor, adding u is free)
- Submodular load balancing problem (Svitkina and Fleischer, 2011)

$$\text{minimize} \quad \max_i \underbrace{|\mathcal{N}(U_i)|}_{\text{memory load}} \quad \text{where } \mathcal{N}(U_i) := \bigcup_{u \in U_i} \mathcal{N}(u)$$

worst client

memory load

neighbors in V

Google

Carnegie Mellon University

Memory

- Submodular load balancing problem (Svitkina and Fleischer, 2011)

$$\text{minimize} \quad \max_i |\mathcal{N}(U_i)| \quad \text{where } \mathcal{N}(U_i) := \bigcup_{u \in U_i} \mathcal{N}(u)$$

- Pick currently worst client
- Pick random subset of candidates in U
- Solve submodular minimization problem with set size penalty
- Unreasonably expensive. Must approximate!

Memory

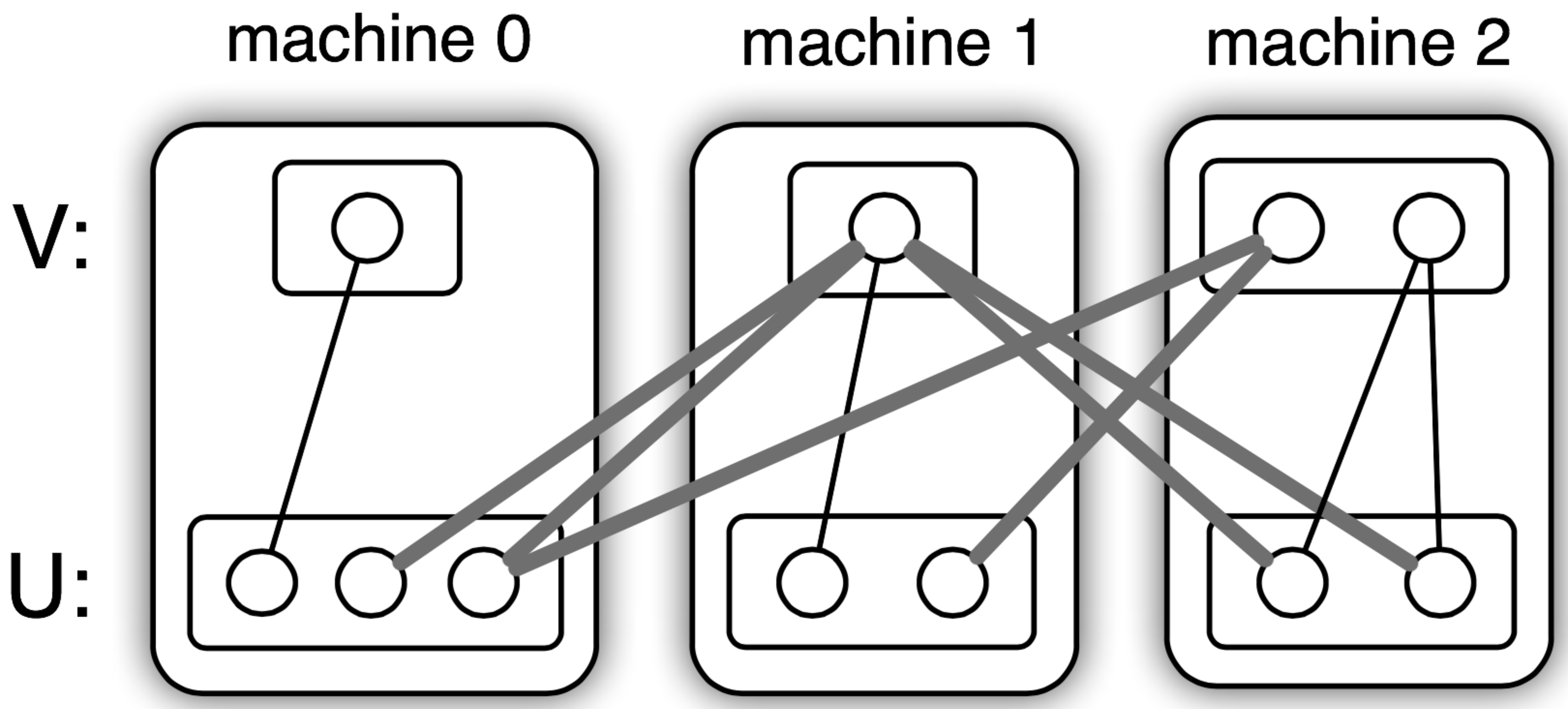
- Submodular load balancing problem (Svitkina and Fleischer, 2011)

$$\text{minimize} \quad \max_i |\mathcal{N}(U_i)| \quad \text{where } \mathcal{N}(U_i) := \bigcup_{u \in U_i} \mathcal{N}(u)$$

- Pick currently worst client i
- Find single best vertex u to add
- Efficient datastructure to cache incremental cost of adding u (many indices are small ints)
- Parallel load balancing in Parameter Server

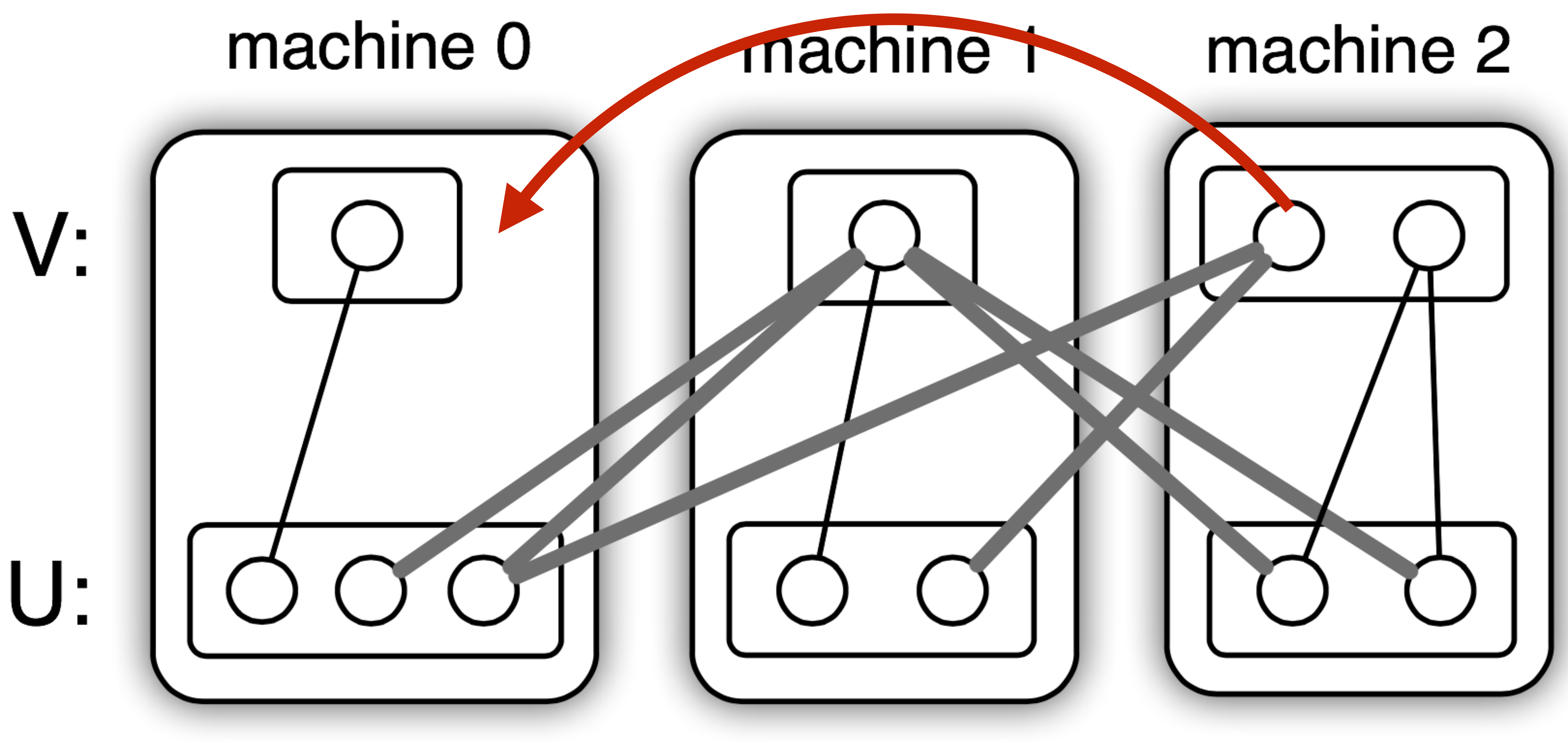
Network

- Put a server on each client
- Communication cost per machine



Network

- Put a server on each client
- Communication cost per machine



Network

- Put a server on each client
- Communication cost per machine

Must cache on j

$$\text{minimize} \quad \max_i |\mathcal{N}(U_i)| - |V_i| + \sum_{j \neq i} |V_i \cap \mathcal{N}(U_j)|$$

Must cache on i

Owned by i

for free on i

Network

- Put a server on each client
- Communication cost per machine

$$\underset{v}{\text{minimize}} \quad \max_i |\mathcal{N}(U_i)| + \sum_j v_{ij} \left[-1 + \sum_{l \neq i} u_{lj} \right]$$

$$\text{subject to } \sum_j v_{ij} = 1 \text{ and } v_{ij} \in \{0, 1\} \text{ and } v_{ij} \leq u_{ij}$$

totally unimodular constraints

Network

- Put a server on each client
- Communication cost per machine

$$\underset{v}{\text{minimize}} \quad \max_i |\mathcal{N}(U_i)| + \sum_j v_{ij} \left[-1 + \sum_{l \neq i} u_{lj} \right]$$

$$\text{subject to } \sum_j v_{ij} = 1 \text{ and } v_{ij} \in \{0, 1\} \text{ and } v_{ij} \leq u_{ij}$$

can find optimal solution

Network

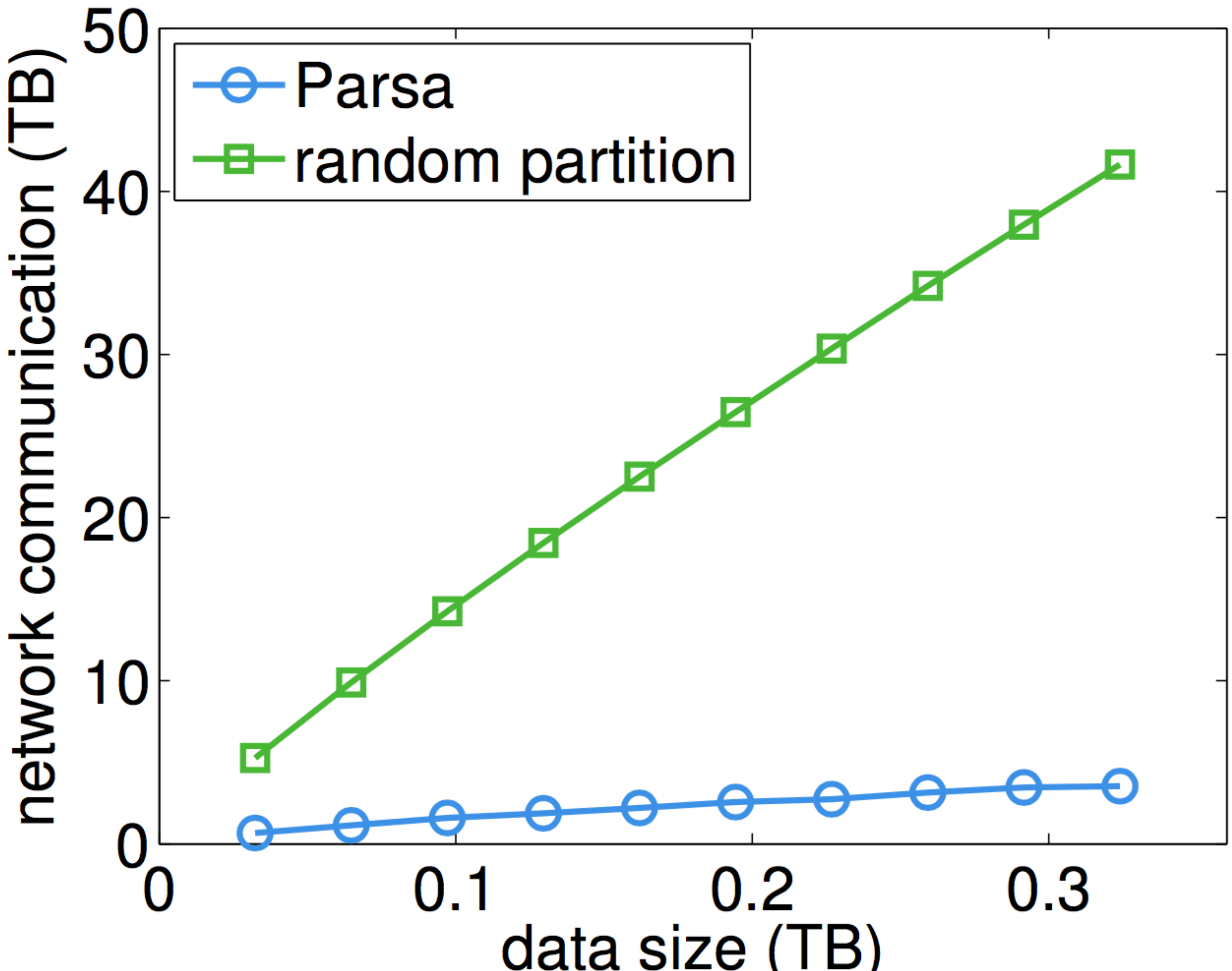
- Put a server on each client
- Communication cost per machine

$$\underset{v}{\text{minimize}} \quad \max_i |\mathcal{N}(U_i)| + \sum_j v_{ij} \left[-1 + \sum_{l \neq i} u_{lj} \right]$$

$$\text{subject to } \sum_j v_{ij} = 1 \text{ and } v_{ij} \in \{0, 1\} \text{ and } v_{ij} \leq u_{ij}$$

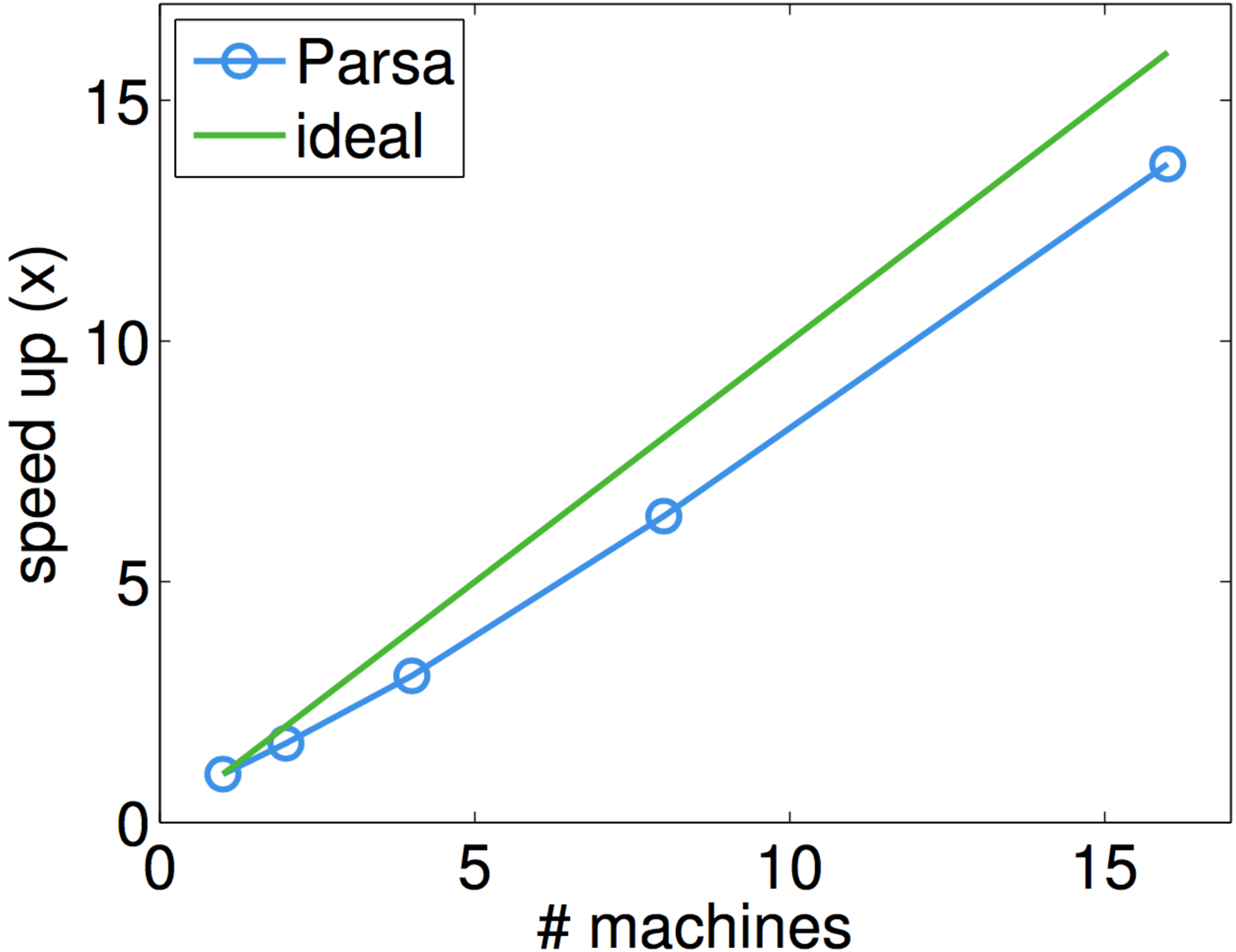
- Iterate over vertices i
- Greedily (re)assign vertex to server

Bandwidth savings



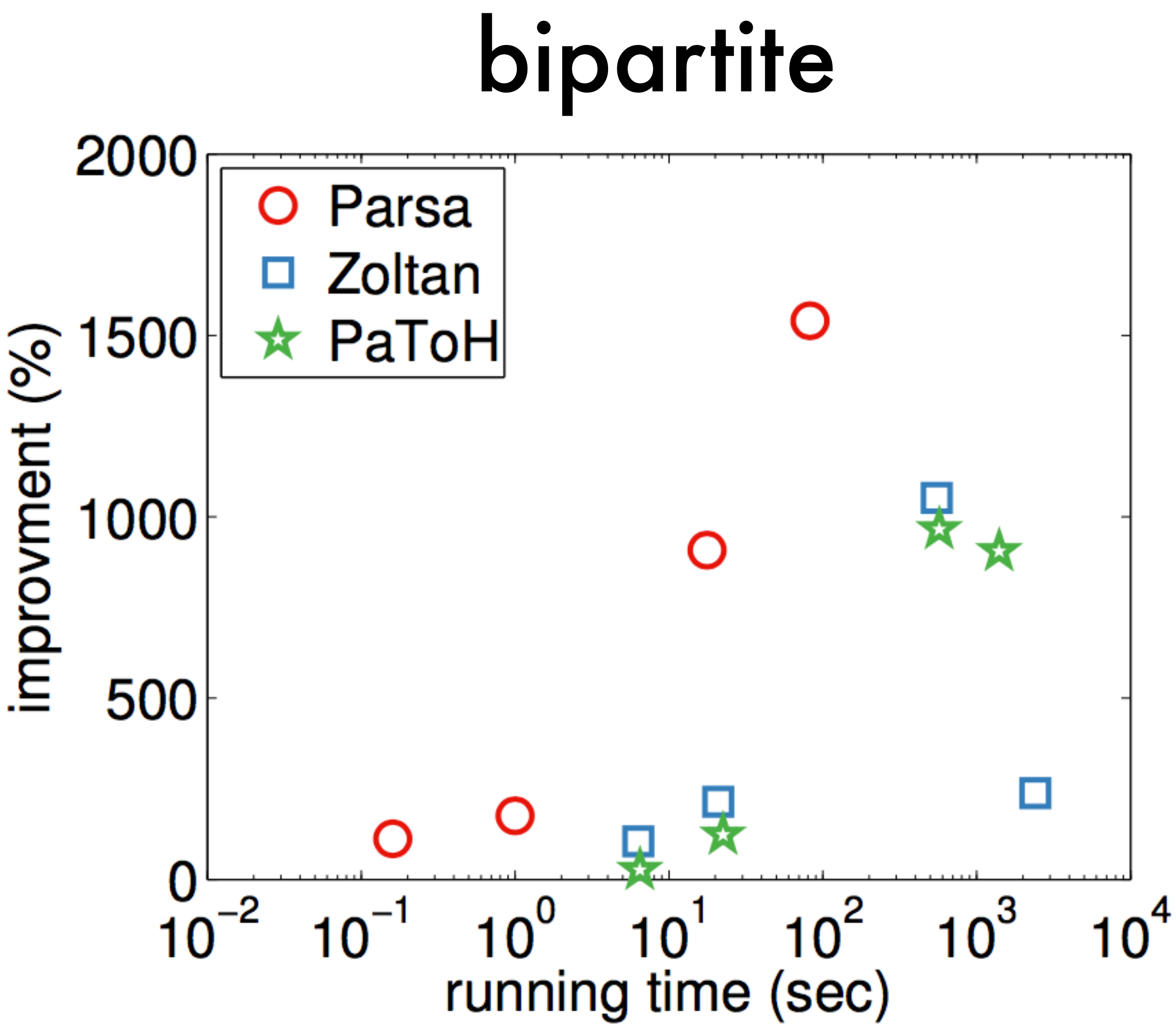
lon University

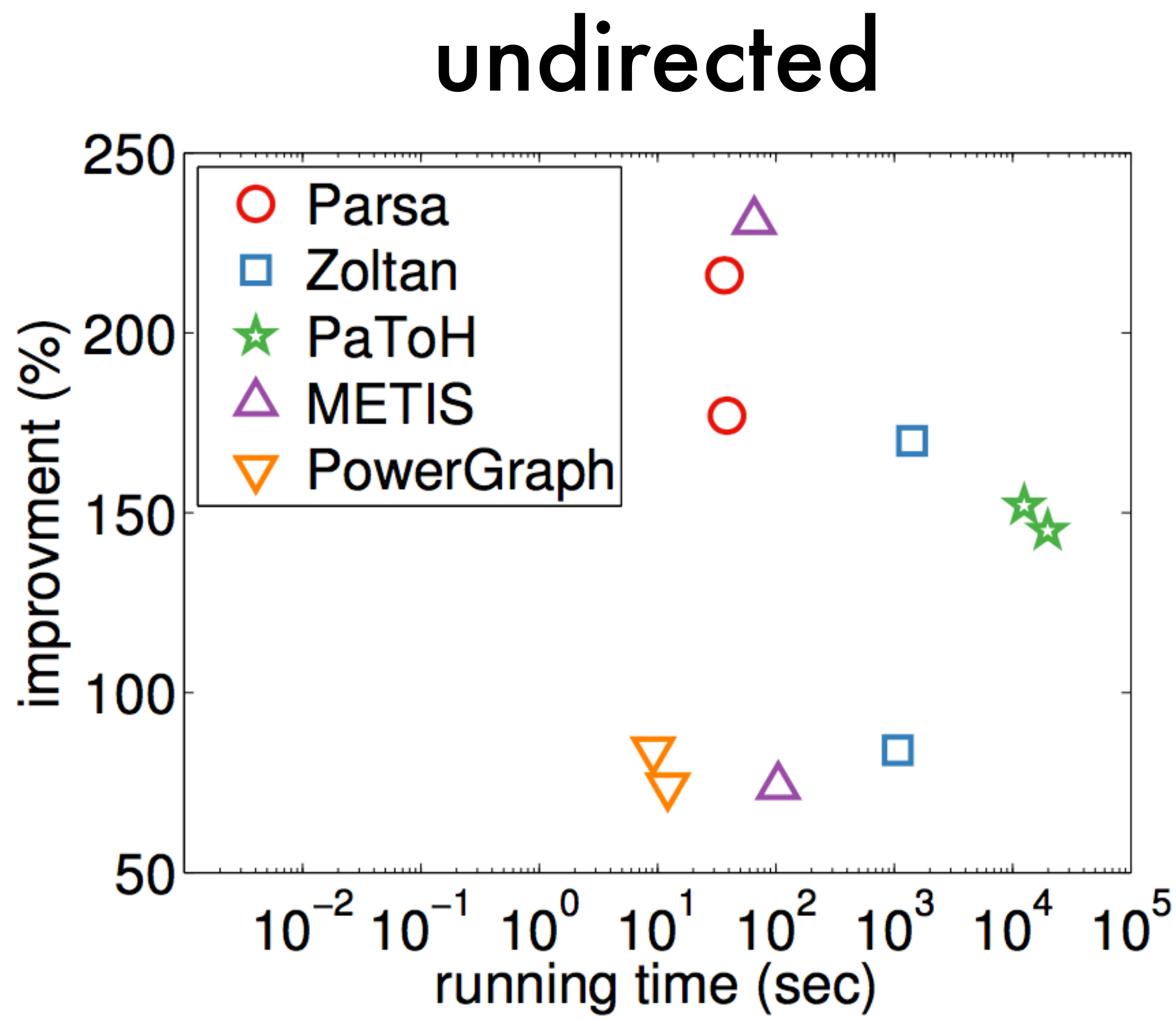
Bandwidth savings



Google
n University

Speed and Accuracy





Outline

- **Motivation**
  Models, hardware
- **Bipartite design**
  Communication, key layout, recovery
- **Efficiency**
  Filters, consistency models
- **Improving the Layout**
  Submodular load balancing
- **Experiments**


Experiments

Logistic Regression

Guinea pig - logistic regression

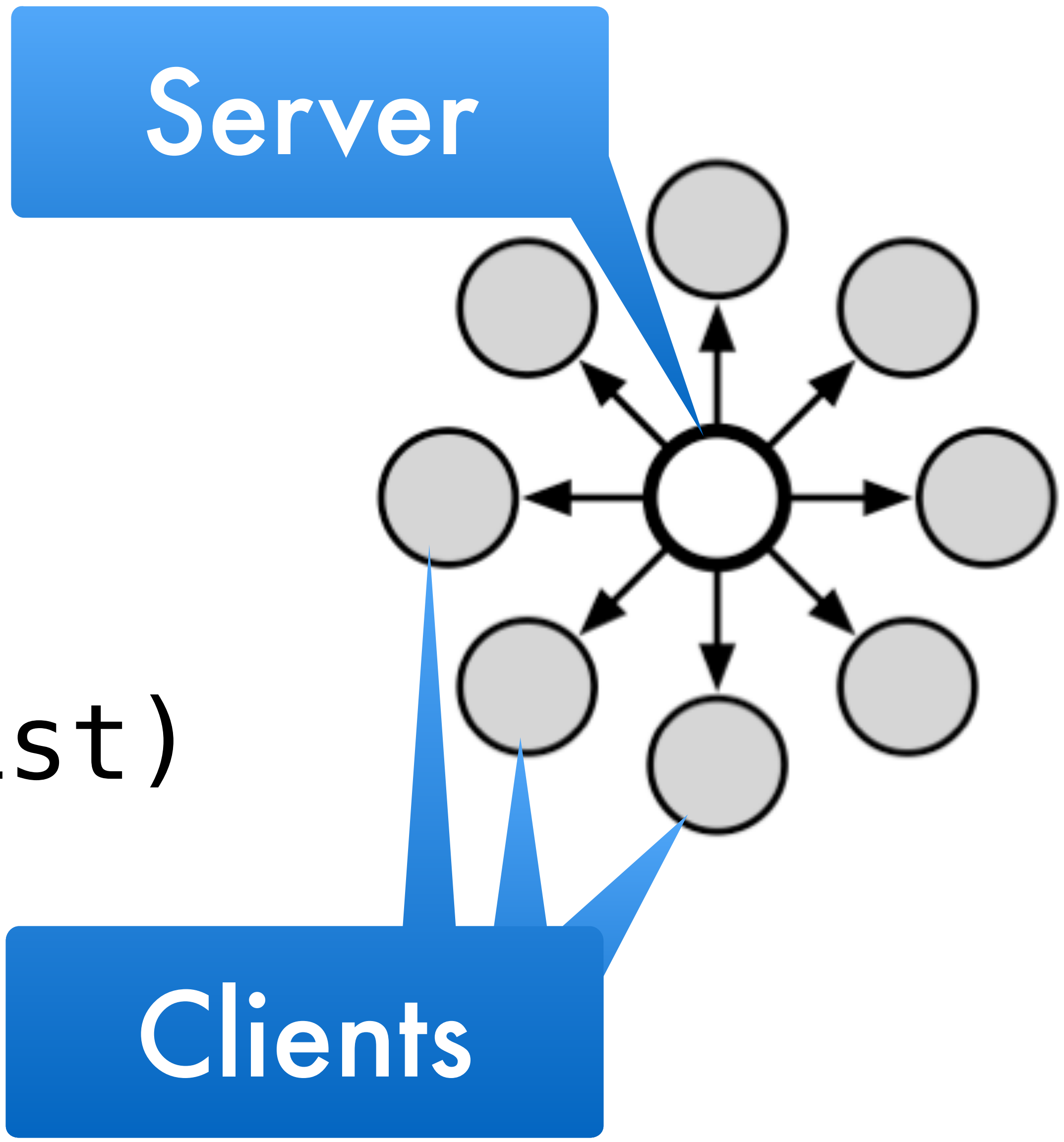
$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1$$

- **Implementation on Parameter Server**

	Method	Consistency	LOC
System-A	L-BFGS	Sequential	10,000
System-B	Block PG	Sequential	30,000
Parameter Server	Block PG	Bounded Delay KKT Filter	300

Recall: Parallel Template

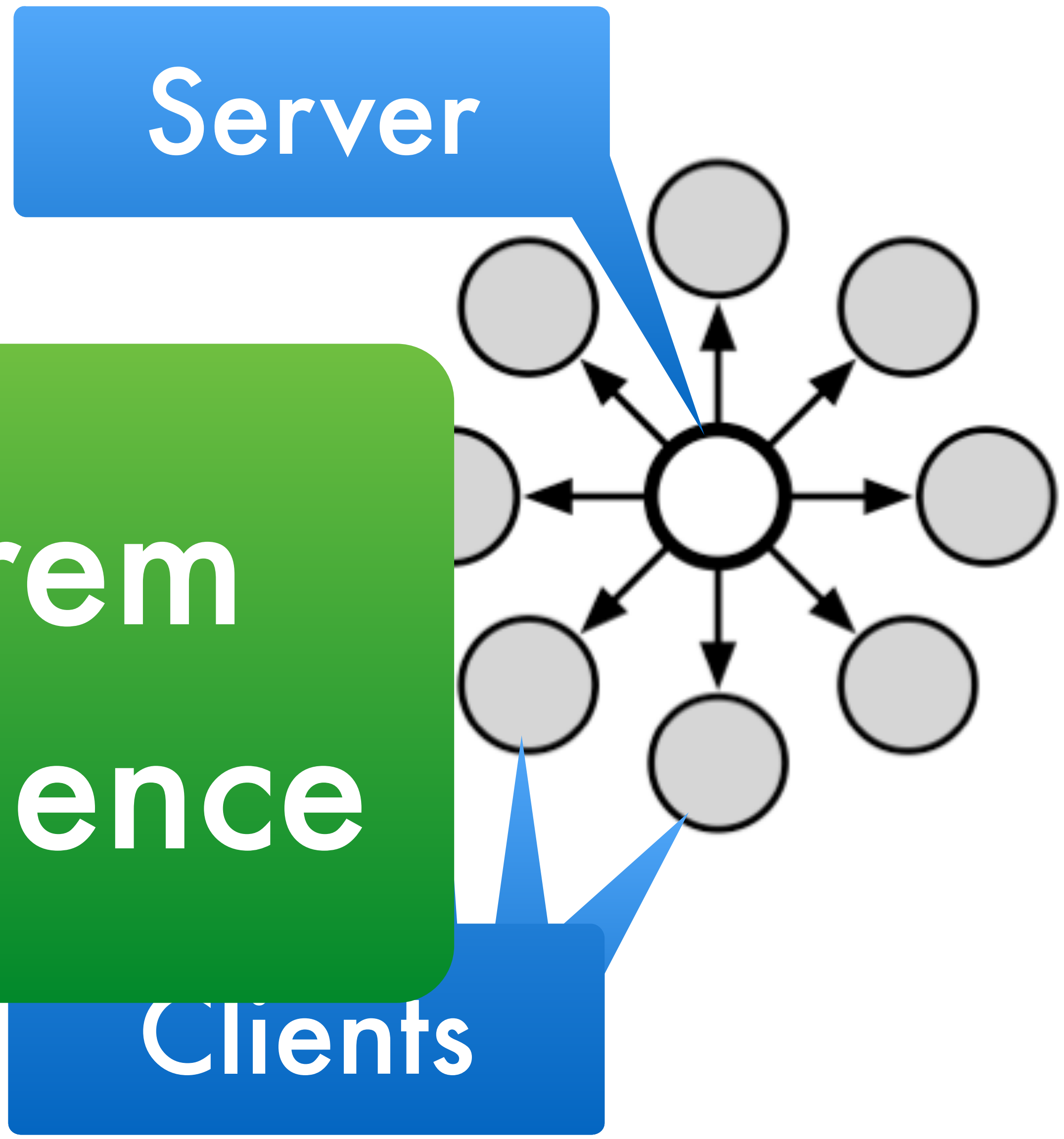
- Compute gradient on (subset of data) on each client
- Send gradient from client to server asynchronously
  `push(key_list,value_list)`
- Proximal gradient update on server per coordinate
- Server returns parameters
  `pull(key_list,value_list)`



Recall: Parallel Template

- Compute gradient on (subset of data) on each client
- Send gra[...] to serve[...] push(ke[...]
- Proxima[...] on server per coordinate
- Server returns parameters pull(key_list,value_list)

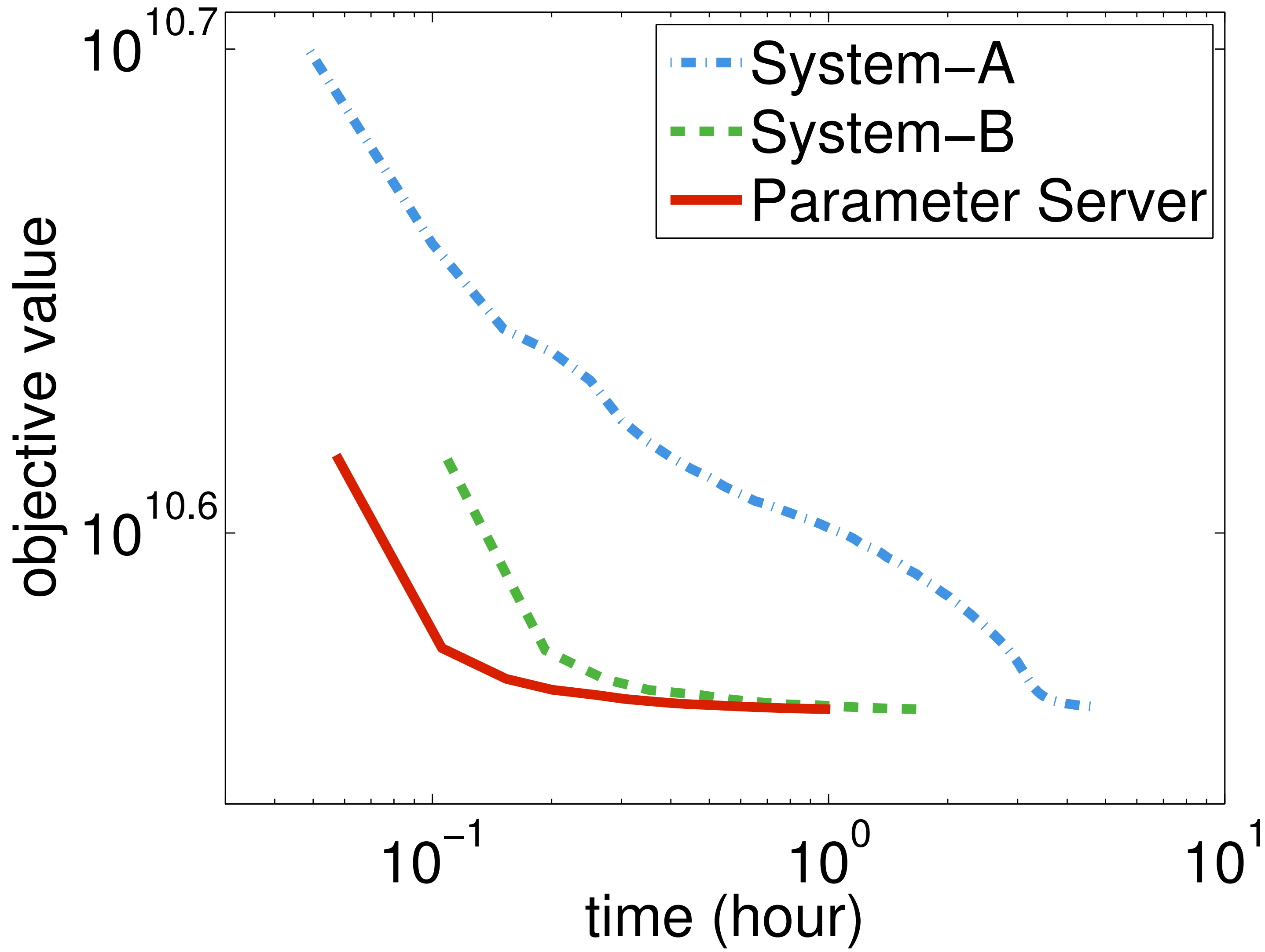
with theorem for convergence



Google

Carnegie Mellon University

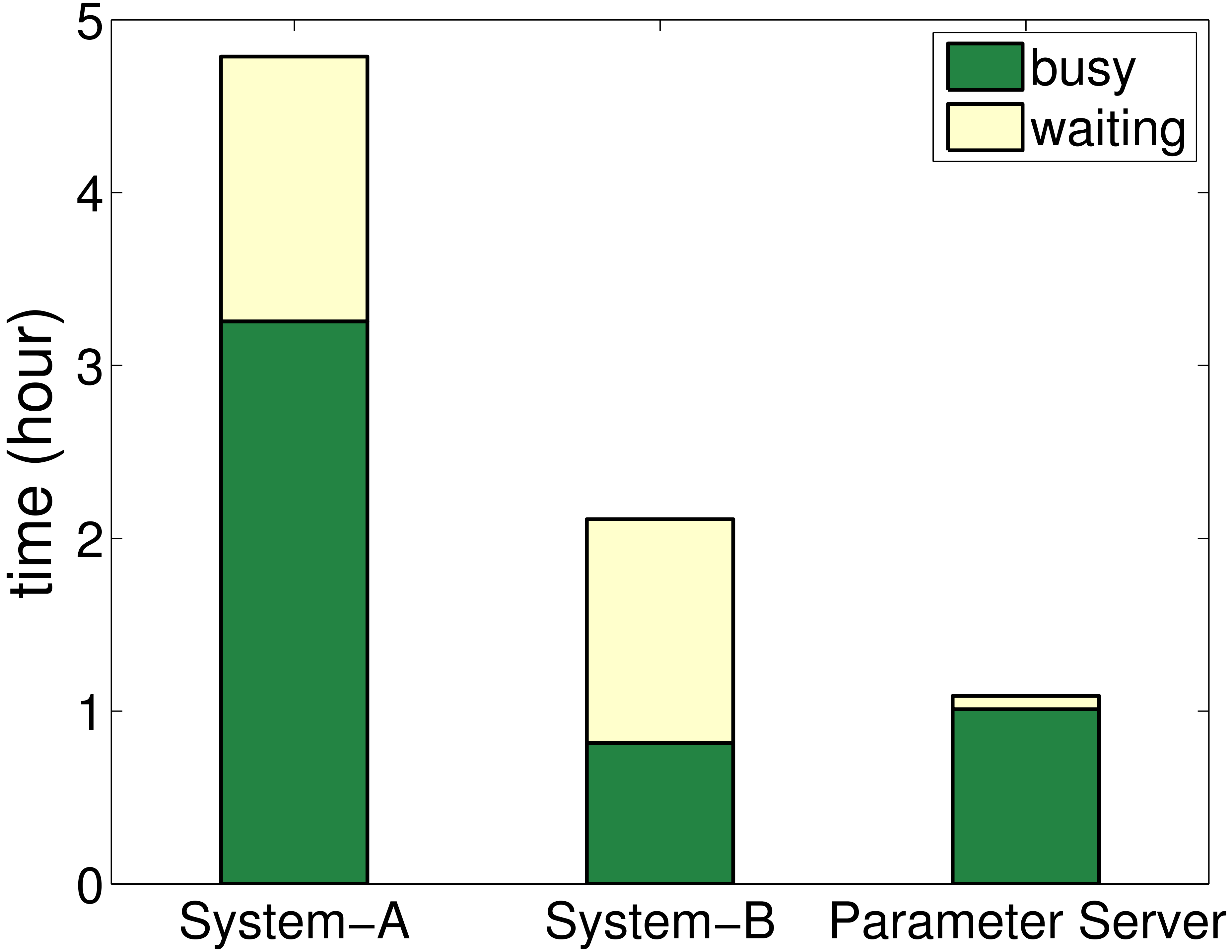
Convergence speed



500TB CTR data
100B variables
1000 machines

- System A and B are production systems at a very large internet company …

Scheduling Efficiency

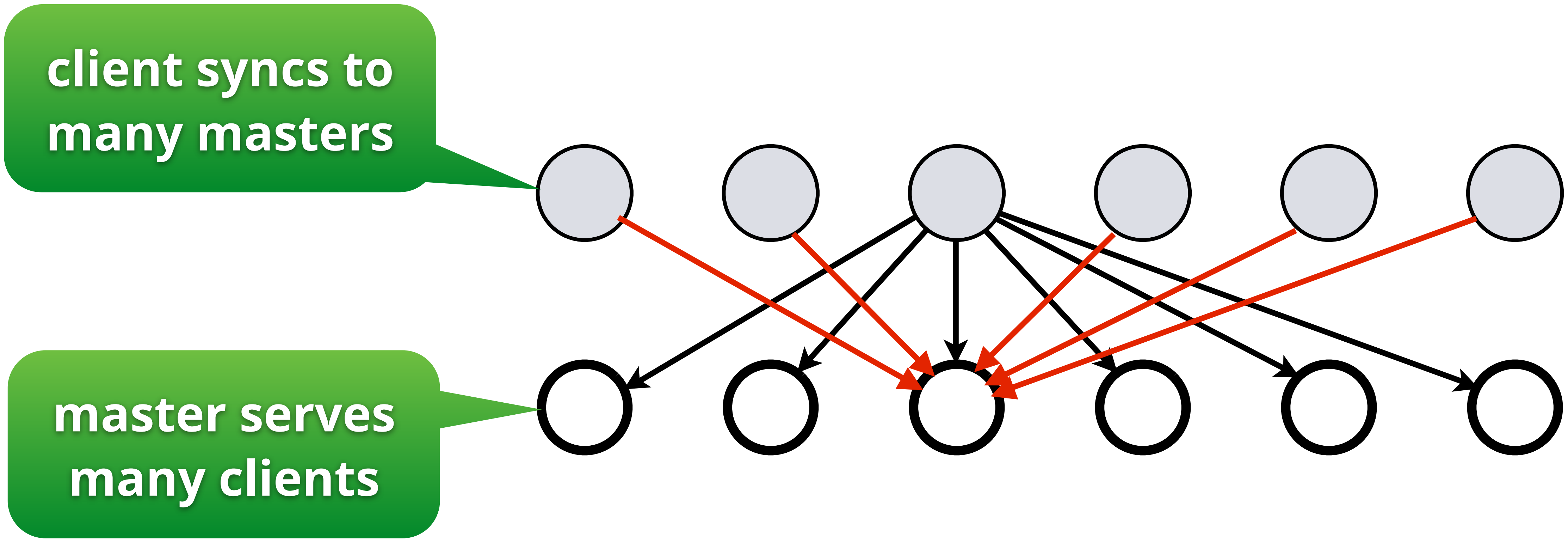


Carnegie Mellon University


Parameter Server as
Stream Processor

Communication Pattern

- Client - Ingest data/query from network (users, CTR, event logger)
- Server - Aggregate sketch (CountMin, SpaceSaver, CounterBraid)



Guinea pig - CountMin Sketch

- Intuition - Bloom Filter with integers (see Muthukrishnan and Cormode, 2005)
- Insert

$$M[h(k,j),j] \leftarrow M[h(k,j),j] + v \text{ for all } j \in \{1, \ldots d\}$$

  Each counter is an upper bound on counts
- Query

$$m(k) \leq \min_j M[h(k,j),j]$$

- Extensions to time series (see Matyusevych, Ahmed, Smola, 2012)

Distributed CountMin Sketch

- Clients only act as data preprocessors
- Shard keys over servers for balancing
- Replication between machines on DHT
- Servers perform simple updates

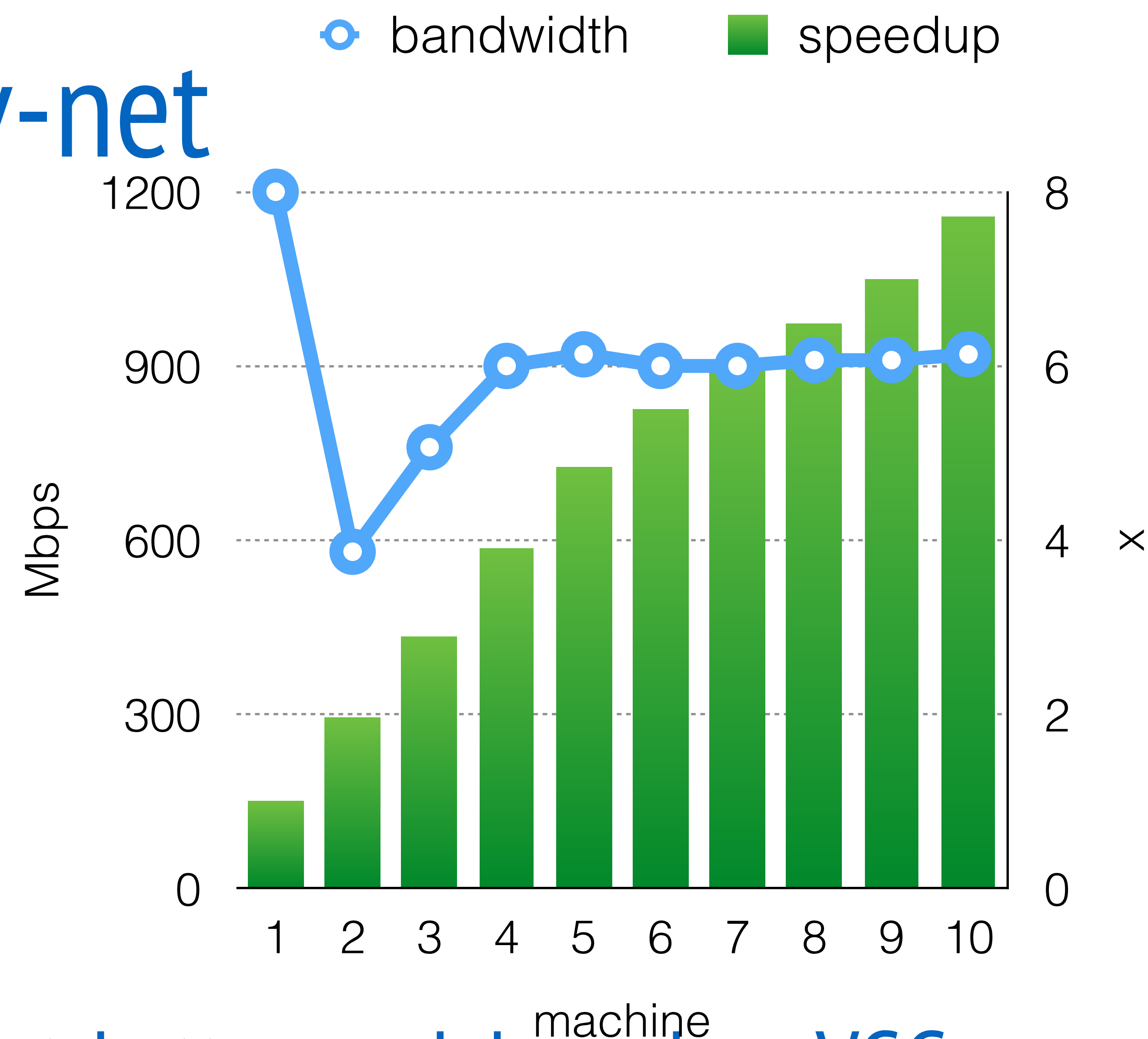
$$M[h(k,j),j] \leftarrow M[h(k,j),j] + v \text{ for all } j \in \{1, \ldots d\}$$

- 15 servers, 40GBit network (dedicated)

Peak inserts per second	1.3 billion
Average inserts per second	1.1 billion
Peak network bandwidth per machine	4.37 GBit/s
Time to recover a failed node	0.8 second

Limited by DRAM Latency

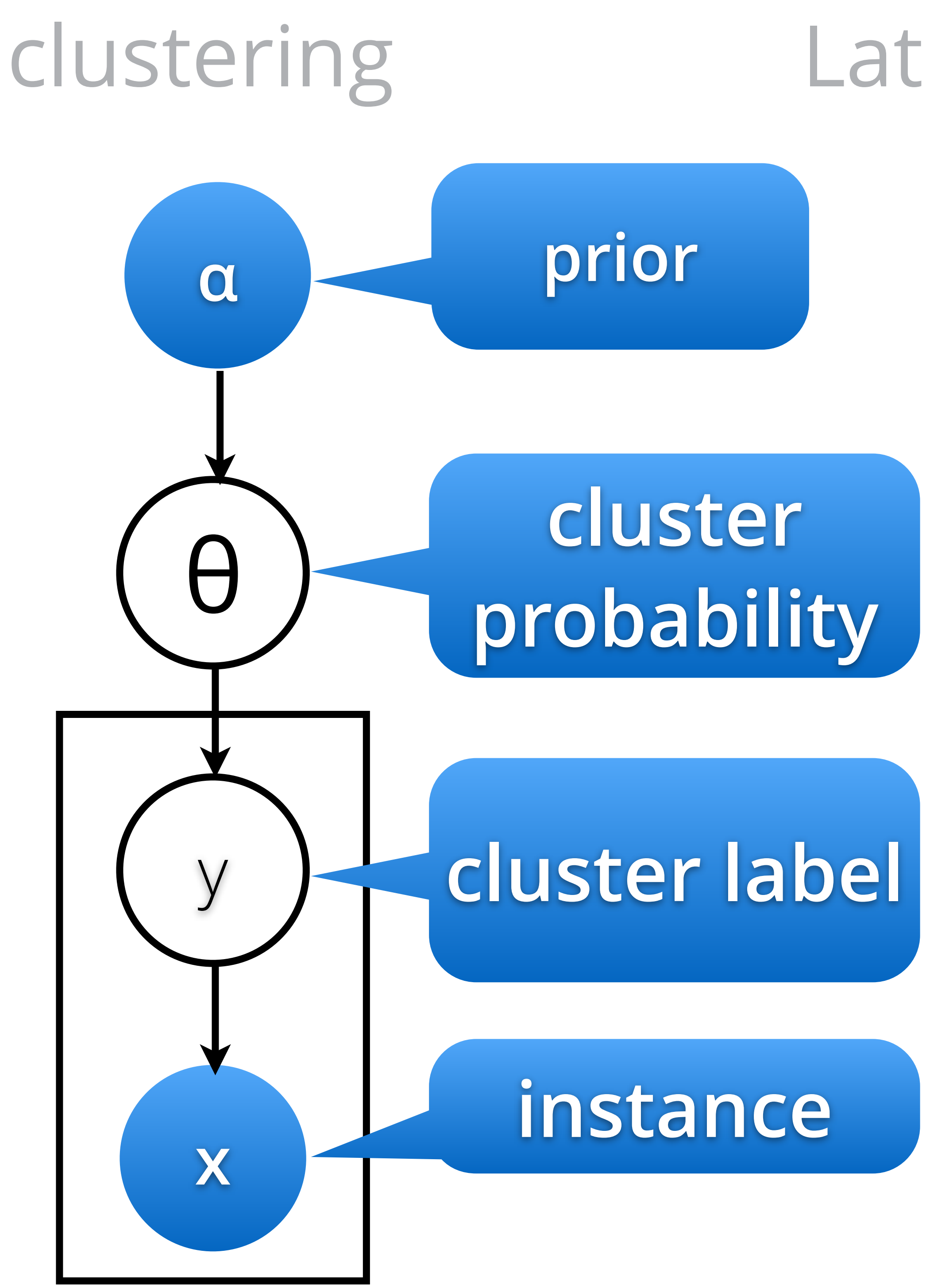
Scalability result on conv-net

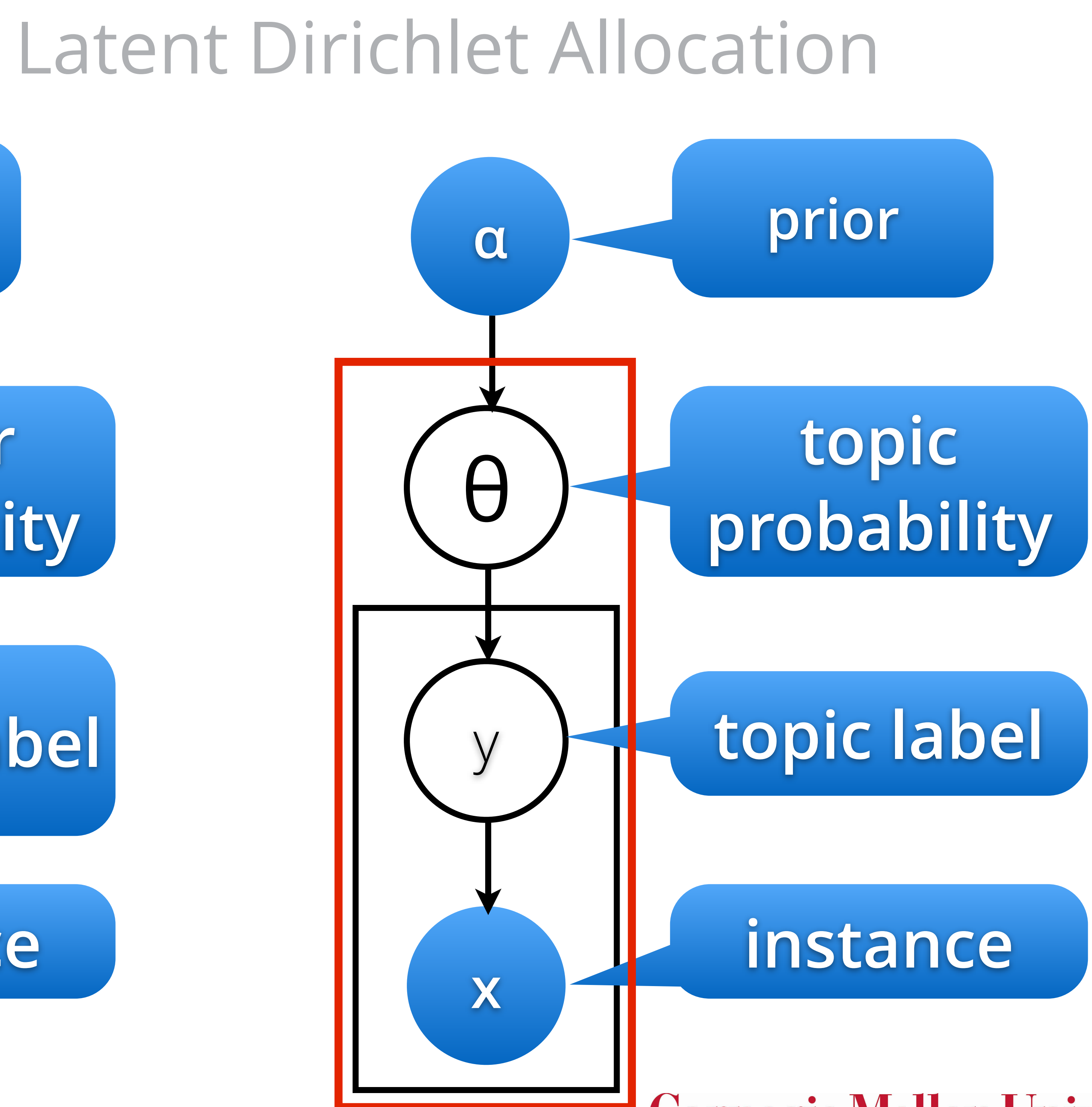


- Two-level parameter server
- CXXNET + AlexNet
- Use ec2 GPU instances
  - reach the hardware limits
- Future work:
  - Alexnet is not the state-of-the-art, better models such as VGG or Googlenet are network friendly
  - More optimization on communication, such as comprising float 32bit->24bit, then the bandwidth required < 900Mbps
  - Our own cluster has 10x larger bandwidth

Models

Clustering & Topic Models





Topics in text

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

Latent Dirichlet Allocation; Blei, Ng, Jordan, JMLR 2003

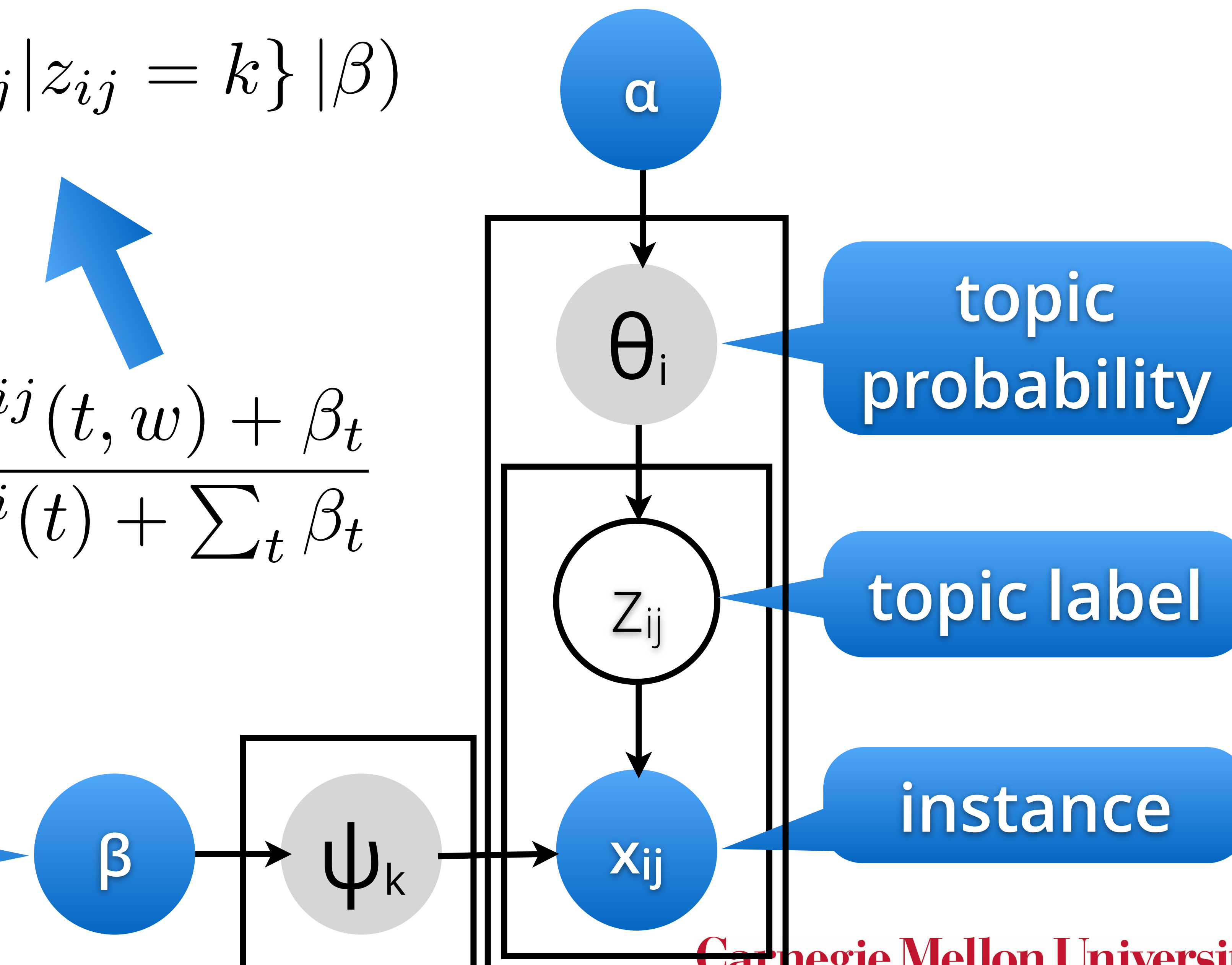
Collapsed Gibbs Sampler

Griffiths & Steyvers, 2005

$$p(z, x|\alpha, \beta)$$

$$= \prod_{i=1}^{m} p(z_i|\alpha) \prod_{k=1}^{k} p(\{x_{ij}|z_{ij} = k\}\,|\beta)$$

$$\frac{n^{-ij}(t,d) + \alpha_t}{n^{-i}(d) + \sum_t \alpha_t} \qquad \frac{n^{-ij}(t,w) + \beta_t}{n^{-i}(t) + \sum_t \beta_t}$$



Collapsed Gibbs Sampler

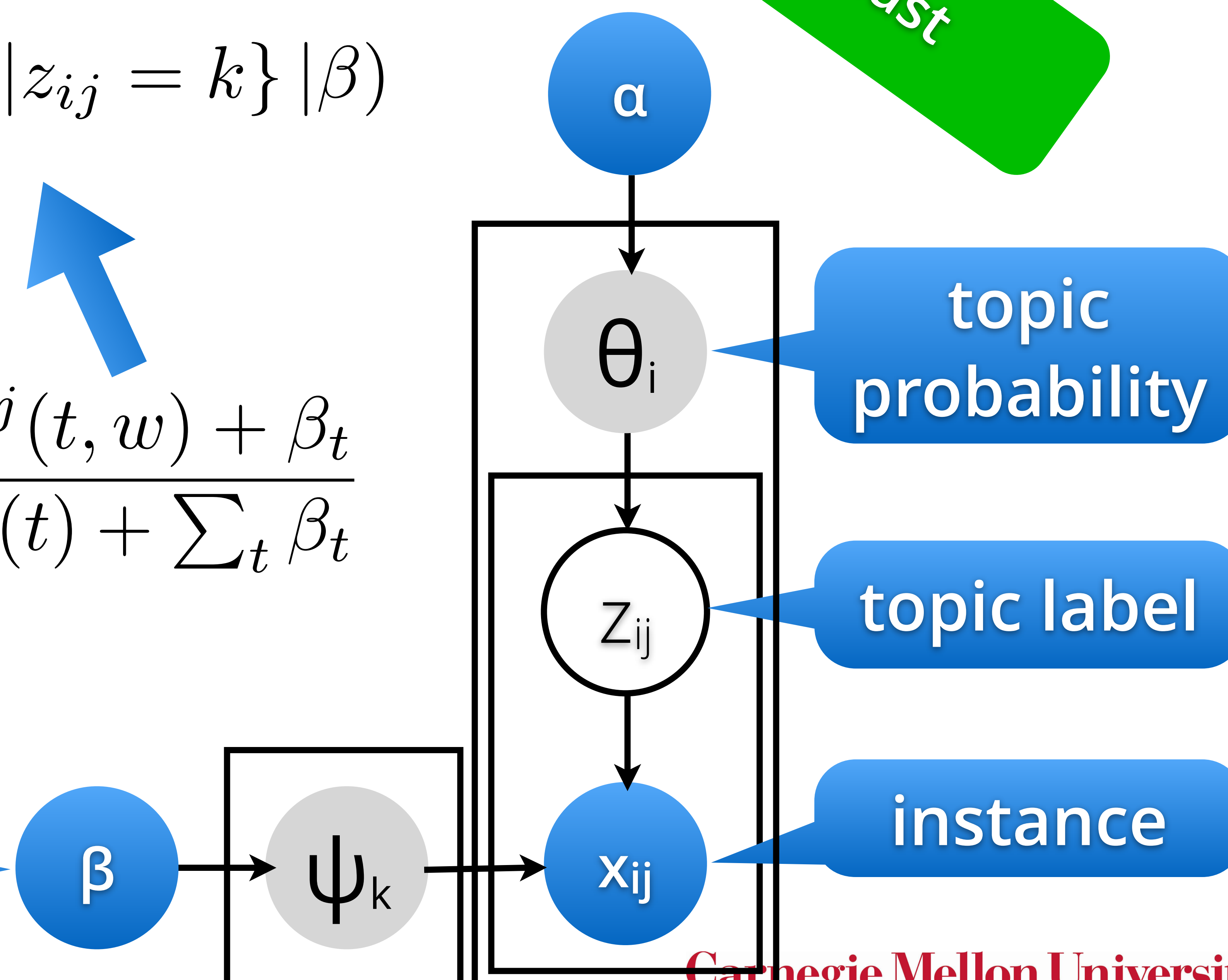
Griffiths & Steyvers, 2005

fast

$$p(z, x|\alpha, \beta)$$

$$= \prod_{i=1}^{m} p(z_i|\alpha) \prod_{k=1}^{k} p(\{x_{ij}|z_{ij} = k\}\,|\beta)$$

$$\frac{n^{-ij}(t,d) + \alpha_t}{n^{-i}(d) + \sum_t \alpha_t} \qquad \frac{n^{-ij}(t,w) + \beta_t}{n^{-i}(t) + \sum_t \beta_t}$$



Gibbs Sampler

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Lock (word,topic) table
      - Update local (document, topic) table
      - Update (word,topic) table
      - Unlock (word,topic) table

this kills parallelism

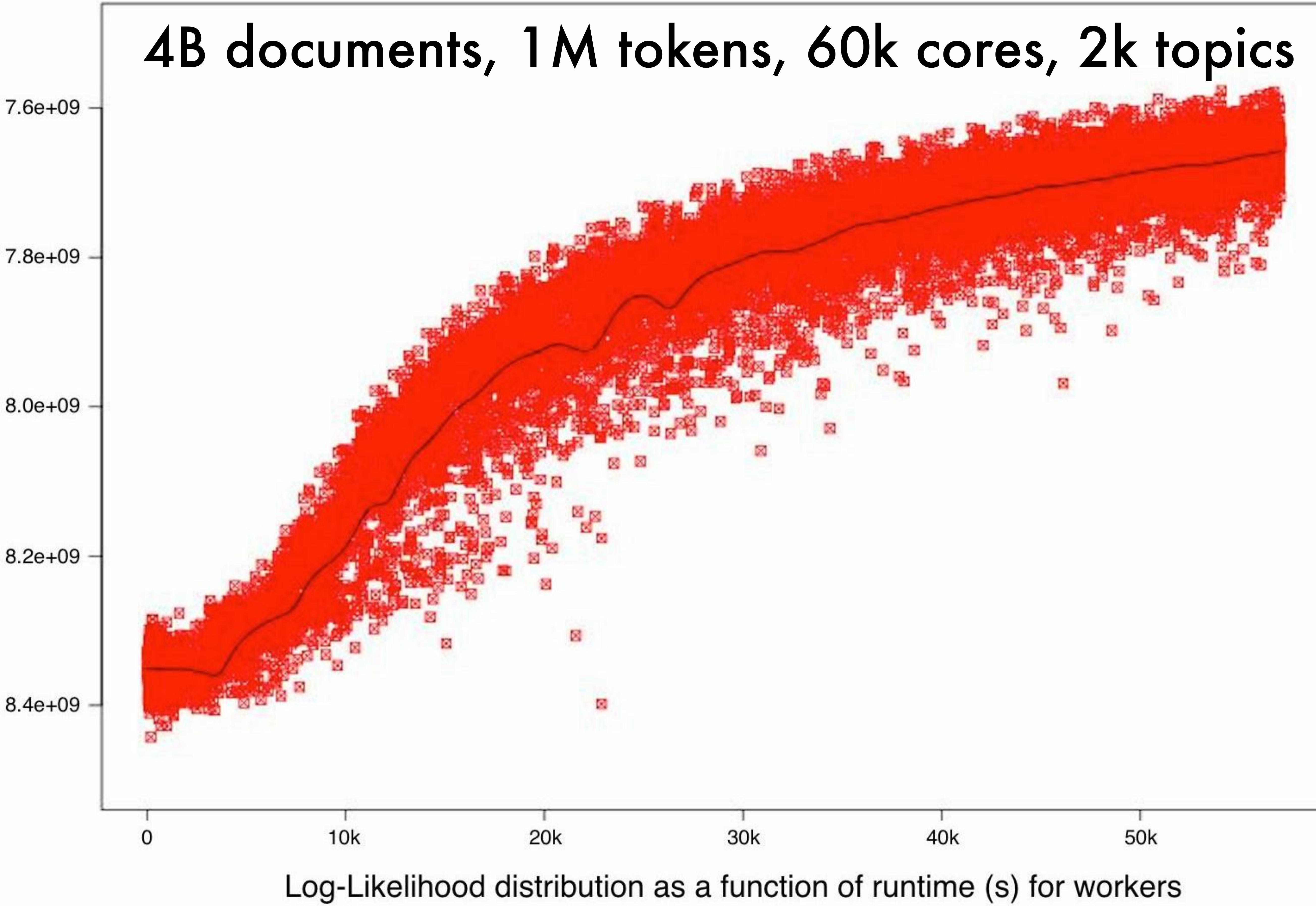
Gibbs Sampler

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Lock local (word,topic) table
      - Update local (document, topic) table
      - Update local (word,topic) table
      - Unlock local (word,topic) table
    - Synchronize local and global tables

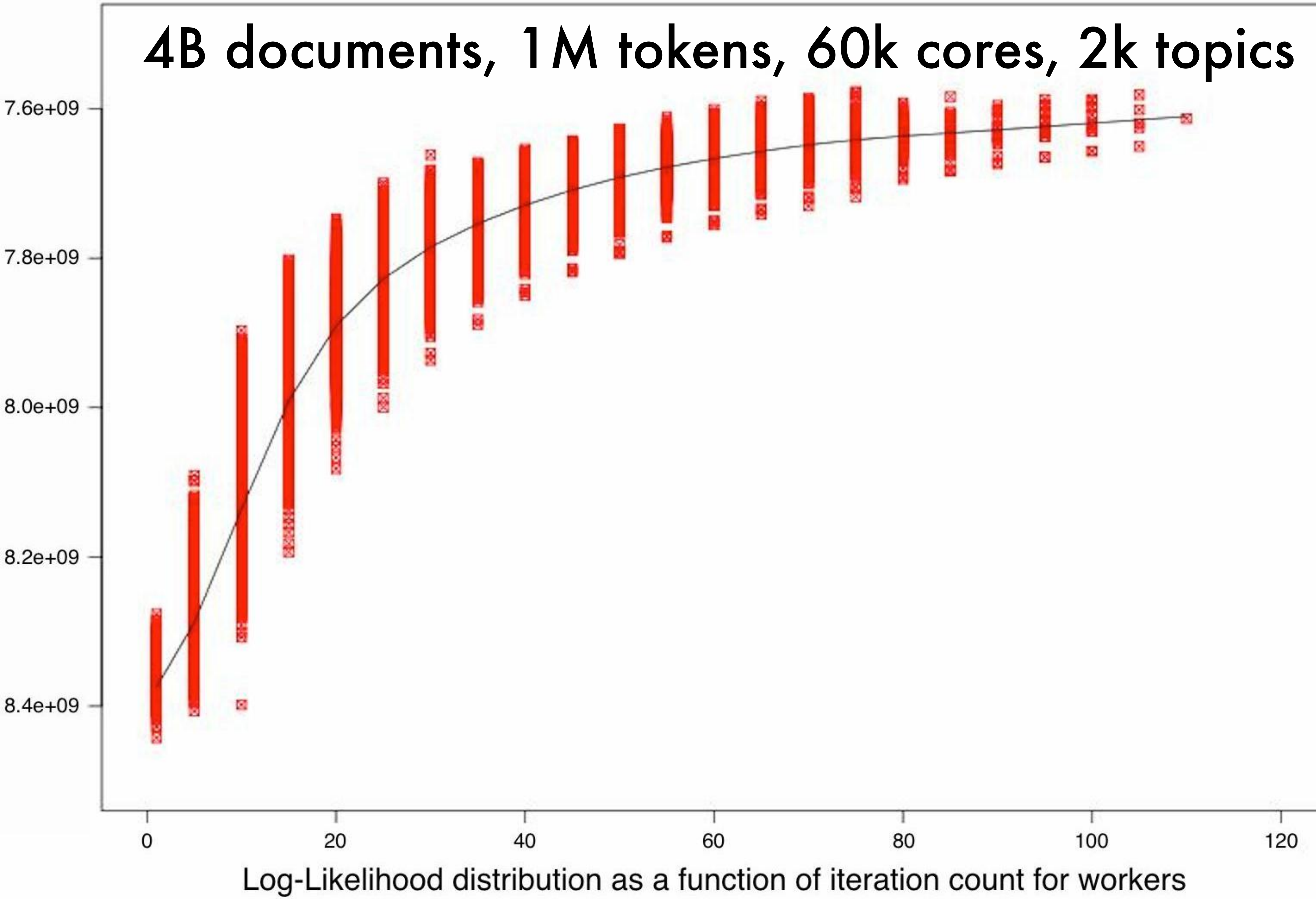
this kills multithreading

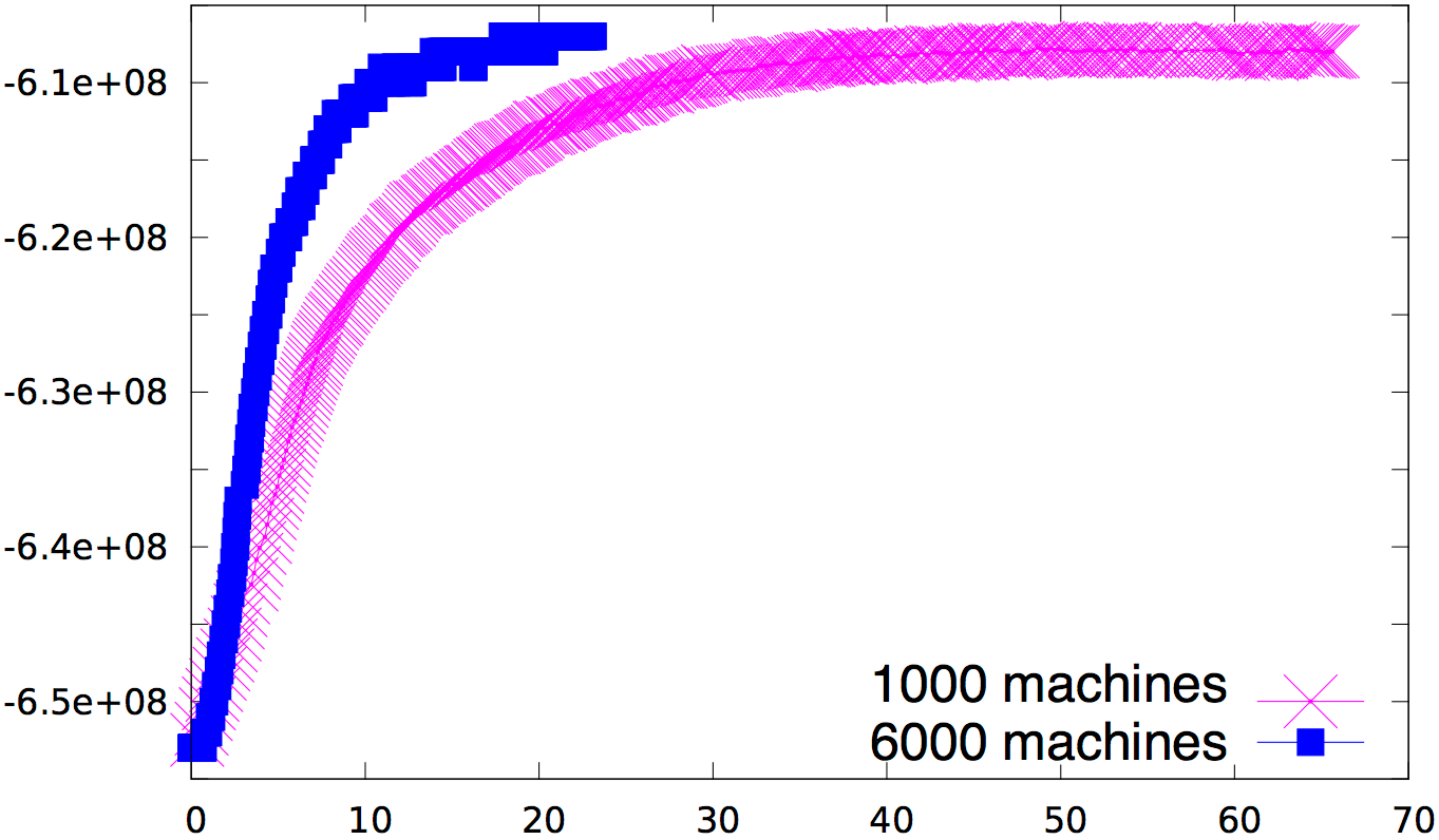
Gibbs Sampler for LDA

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Update local (document, topic) table
      - Generate local update message
    - Update local table
      - Lock local (word,topic) table
      - Update local (word,topic) table
      - Unlock local (word,topic) table
    - Synchronize local and global tables

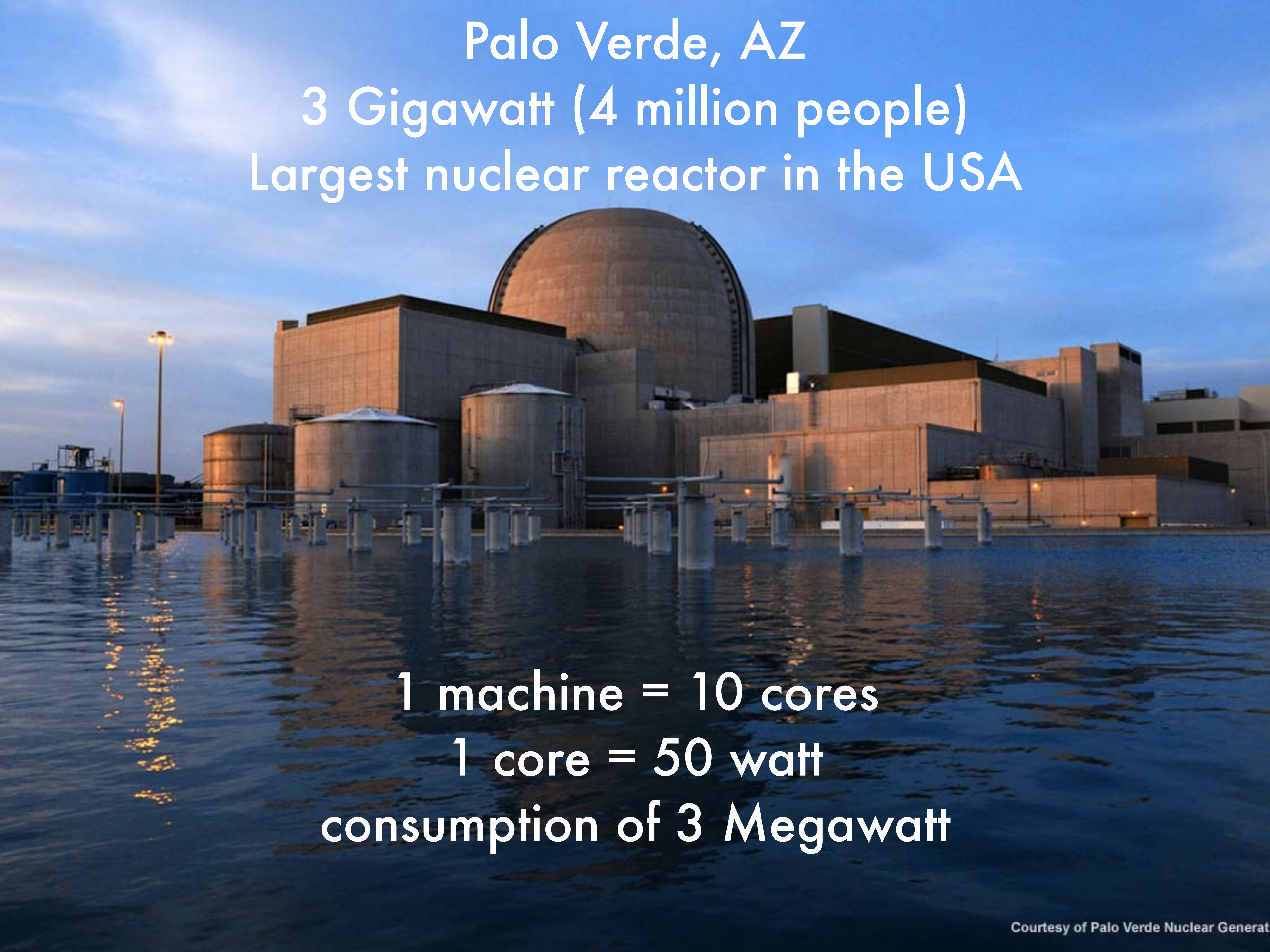

4B documents, 1M tokens, 60k cores, 2k topics
7.6e+09
7.8e+09
8.0e+09
8.2e+09
8.4e+09
0
10k
20k
30k
40k
50k
Log-Likelihood distribution as a function of runtime (s) for workers

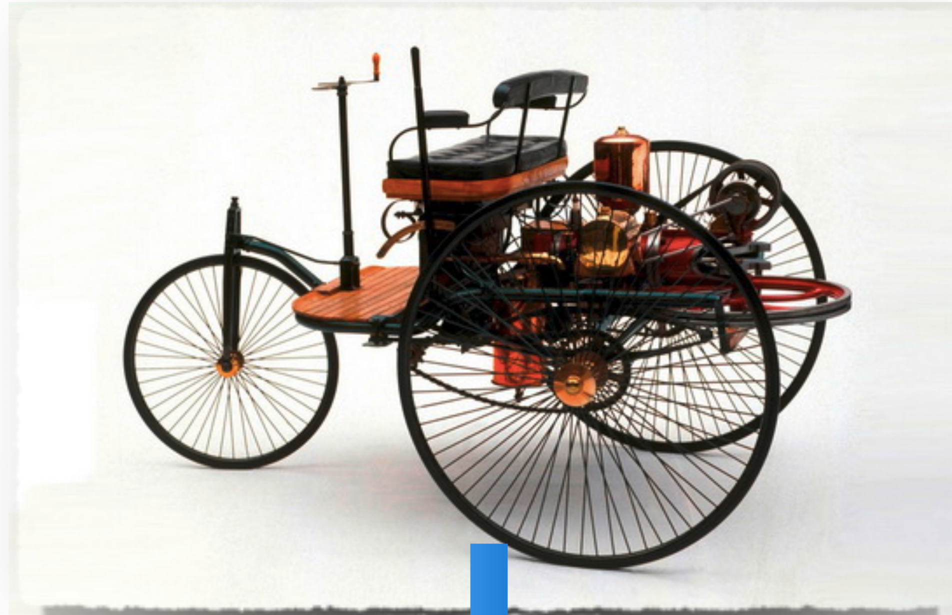
4B documents, 1M tokens, 60k cores, 2k topics




-6.1e+08
-6.2e+08
-6.3e+08
-6.4e+08
-6.5e+08
0
10
20
30
40
50
60
70
1000 machines
6000 machines


Palo Verde, AZ
3 Gigawatt (4 million people)
Largest nuclear reactor in the USA
Courtesy of Palo Verde Nuclear Generat


Palo Verde, AZ
3 Gigawatt (4 million people)
Largest nuclear reactor in the USA
1 machine = 10 cores
1 core = 50 watt
consumption of 3 Megawatt
Courtesy of Palo Verde Nuclear Generat



Convenience

Performance

parameterserver.org

blog.smola.org @smolix